

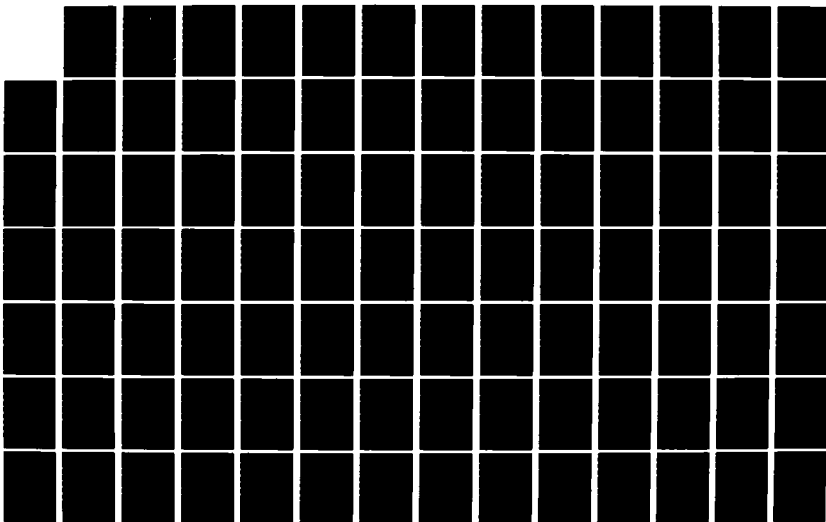
AD-A124 851

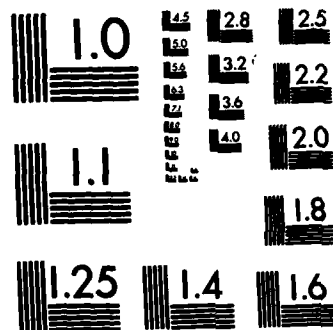
ISOLATED WORD RECOGNITION USING FUZZY SET THEORY(U) AIR 1/4
FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING G J MONTGOMERY DEC 82 AFIT/GE/EE/82D-74

UNCLASSIFIED

F/G 5/8

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

1



AD A124811



DTIC FILE COPY

DTIC
ELECTE
FEB 24 1983
S E D

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

This document has been approved
for public release and sale; its
distribution is unlimited.

83 02 022 05

AFIT/GE/EE/82D-74

ISOLATED WORD RECOGNITION
USING FUZZY SET THEORY

THESIS

AFIT/GE/EE/82D-74 Gerard J. Montgomery
2nd Lt USAF

DTIC
S
E

Approved for public release; distribution unlimited.

ISOLATED WORD RECOGNITION
USING FUZZY SET THEORY

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology

Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by
Gerard J. Montgomery, B.S.
2nd Lt USAF
Graduate Electrical Engineering
December 1982

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	



Approved for public release; distribution unlimited.

Preface

This project has been motivated by the research of Dr. Matthew Kabrisky, Professor of Electrical Engineering, Air Force Institute of Technology. This research produced an isolated word recognition algorithm capable of identifying the word spoken given the highly variable output of an acoustic process. This report presents the theory, results, and computer programs for this research effort, and provides recommendations for future research.

I would like to thank my advisors, Major H. Carter, Lt Rob Milne, and Major Larry R. Kizer for their guidance during this project; and give special thanks to Dr. Matthew Kabrisky whose insight, enthusiasm, and assistance contributed greatly to the success of this research.

My utmost appreciation is to my wife, Vicki, whose understanding, and support made this thesis possible.

Gerard J. Montgomery

Contents

	<u>Page</u>
Preface	ii
List of Figures	v
List of Tables	viii
Abstract	ix
I. Introduction	1
Background	1
Problem	3
Scope	4
General Approach	5
Sequence of Presentation	6
II. Acoustic Output	7
Output Format	7
Output Accuracy	13
III. Fuzzy Set Theory	20
Introduction	20
Fuzzy Set Theory	20
Algebraic Properties	21
Decision Making Using Fuzzy Sets	24
General Remarks	28
IV. Word Recognition Algorithm	29
Introduction	29
Fuzzy Statistics	30
Vector Scoring Algorithm	41
Transition Algorithm	47
Total Word Score	51
Computation of Fuzzy Statistics	53
V. Determination of Word Representations	57
VI. Optimization of the Fuzzy Variables	64
VII. Results	69
VIII. Conclusions	80

Contents

	<u>Page</u>
IX. Recommendations	82
Bibliography	85
Appendix A: Fuzzy Variable Limits	87
Appendix B: Recommended Approach for Designing a Speech Understanding System	88
Appendix C: Computer Program Structure Charts	114
Appendix D: Computer Programs	135
VITA	283

List of Figures

<u>Figure</u>		<u>Page</u>
1	Block Diagram of Speech Recognition Processes	2
2	Acoustic Output for Word "Zero" Spoken by GJM	11
3	Acoustic Output for Word "Zero" Spoken by DCD	14
4	Acoustic Output for Word "Zero" Spoken by JEF	15
5	Acoustic Output for Word "Zero" Spoken by LMR	16
6	Acoustic Output for Word "Zero" Spoken by MJK	17
7	Acoustic Output for Word "Zero" Spoken by RLC	18
8	Performance Data Maintained for Each Word (Data Shown is for Word "Zero")	33
9	Example of the Overall and Word Fuzzy Variables	34
10	Deletion Statistics Generated for the Word "Zero"	34
11	Substitution Statistics for Word "Zero" for Prototype Phonemes	35
12	Insertion Statistics for Word "Zero" for Prototype Phonemes	37
13	Transition Diagram Indicating the Possible Transitions That Occur Between Word Phonemes	48
14	Vector Results Obtained by Applying Recognition Algorithm to the Speech File Shown in Figure 2 Using the Data Shown in Figures 8 Through 12	54

<u>Figure</u>		<u>Page</u>
15	Steps Required to Initialize Recognition Algorithm for an Experiment	70
16	Steps Required to Obtain Recognition Results After the Steps in Figure 16 Have Been Executed	72
17	Vocabulary and Corresponding Word Representations	76
18	Results of Experiment 2	78
19	Block Diagram of a Speech Understanding System	89
20	Effects of Average Branching Factor on Recognition Error Rate	94
21	Block Diagram of a Possible Computer Architecture for Implementing the Word Recognition Algorithm	101
22	Possible Algorithm for Generating Sentence Hypotheses	104
23	Results of Applying Step 1 of Figure 24	105
24	Tree Resulting From the Application of Step 2 to the Data Shown in Figure 23	106
25	Results of Applying Steps 3 Through 6 of Figure 24 with X Equal to Three	107
26	Legend for the Structure Charts	115
27	Structure Chart 0: Program Learnword	116
28	Structure Chart 1.0: Collect Statistics on a Speech File	117
29	Structure Chart 2.0: Determine Word Spoken. Procedure "ISORECOG"	118
30	Structure Chart 3.0: Determine Word Phoneme Representation. Procedure "DETPHREP"	119

<u>Figure</u>		<u>Page</u>
31	Structure Chart 4.0: Manually Change Word Phoneme Representation. Procedure "CHNGEREP"	120
32	Structure Chart 5.0: Change Fuzzy Variables. Procedure "CHNGEFUZ"	121
33	Structure Chart 6.0: Run an Automated Procedure. Procedure "AUTOMAT"	122
34	Structure Chart 6.1: Store Speech Files. Procedure "STORE SPEECH"	123
35	Structure Chart 6.2: Automatic Word Recognition. Procedure "RECOGALL"	124
36	Structure Chart 6.3: Generate Multiple Word Statistics. Procedure "INITALL"	125
37	Structure Chart 6.4: Determine Multiple Word Representations. Procedure "DETPHALL"	126
38	Structure Chart 6.5: Optimize the Word Fuzzy Variables. Procedure "FUZZYOPT"	127
39	Structure Chart 6.5.1: Optimize Word Variables	128
40	Structure Chart 6.5.2: Compute Word Recognition Results. Procedure "OPTRECOG"	129
41	Structure Chart 6.6: Change Optimum Fuzzy Variables. Procedure "CHNGEOPT"	130
42	Structure Chart A.0: Score Word and Determine Errors. Procedure "SCOREWORD"	131
43	Structure Chart B.0: Generate Word Statistics. Procedure "INITSTAT"	132
44	Structure Chart C.0: Derive an Initial Representation. Procedure "AUTODET"	133
45	Structure Chart D.0: Score All Stored Files for a Word. Procedure "WORDRECOG"	134

List of Tables

<u>Table</u>		<u>Page</u>
I	Set of Phonemes Extracted From Seelandt's Speech	8

Abstract

↓
A unique algorithm was developed to recognize isolated words given the output of an extremely variable feature extraction process. Because of the high error rate of the acoustic processor, it was necessary to rely on the consistency of the sequences of phonemes, and the errors that typically occur for a given word to determine the word spoken. This was accomplished by generating error statistics and phoneme representations for each word in the vocabulary using a set of training speech files. The top five phoneme choices provided by the acoustic processor, and information indicating the accuracy of each choice, for each time segment of speech was implemented. Fuzzy set theory was used to combine this information with the error information obtained from the training files to determine the word spoken. ↙ Because of the number and types of errors present in the acoustic output, the algorithm produces a word score indicating the plausibility of a word being spoken regardless of the number of errors that occurred. To recognize a speech file, a score is generated for each word in the vocabulary, and the word with highest score is chosen as the word that was spoken. The number of operations required to make a decision increases linearly as the size of the vocabulary increases, and the algorithm can readily be adapted for real time speech recognition. The algorithm can also be

applied to a large range of other problems where decisions must be based on data containing numerous errors.

I. Introduction

In the past 30 years there has been a significant amount of research conducted in the area of automatic speech recognition. The reason so much attention has been given to this area is that the possible applications for a speech recognition machine appear to be unlimited. The list of possible applications include the military, industry, office, school, and home applications; and involve almost any area of human endeavor in which there is a human-machine interface. Although there has been some promising results in this area of research, automatic speech recognition remains a distant goal.

Background

There are many reasons why there are currently no efficient automatic speech recognition systems, but perhaps the major reason is that there is a lack of appropriate decision algorithms. Speech recognition, as with any other pattern recognition problem, can be separated into two processes: a feature extraction process, and a decision process. Figure 1 illustrates these processes for speech recognition. First, features are extracted from the speech, and then these features are input into the decision process which determines the words that were spoken. Because of the inability of present decision algorithms to efficiently determine which words spoken, given a set of features containing errors, many

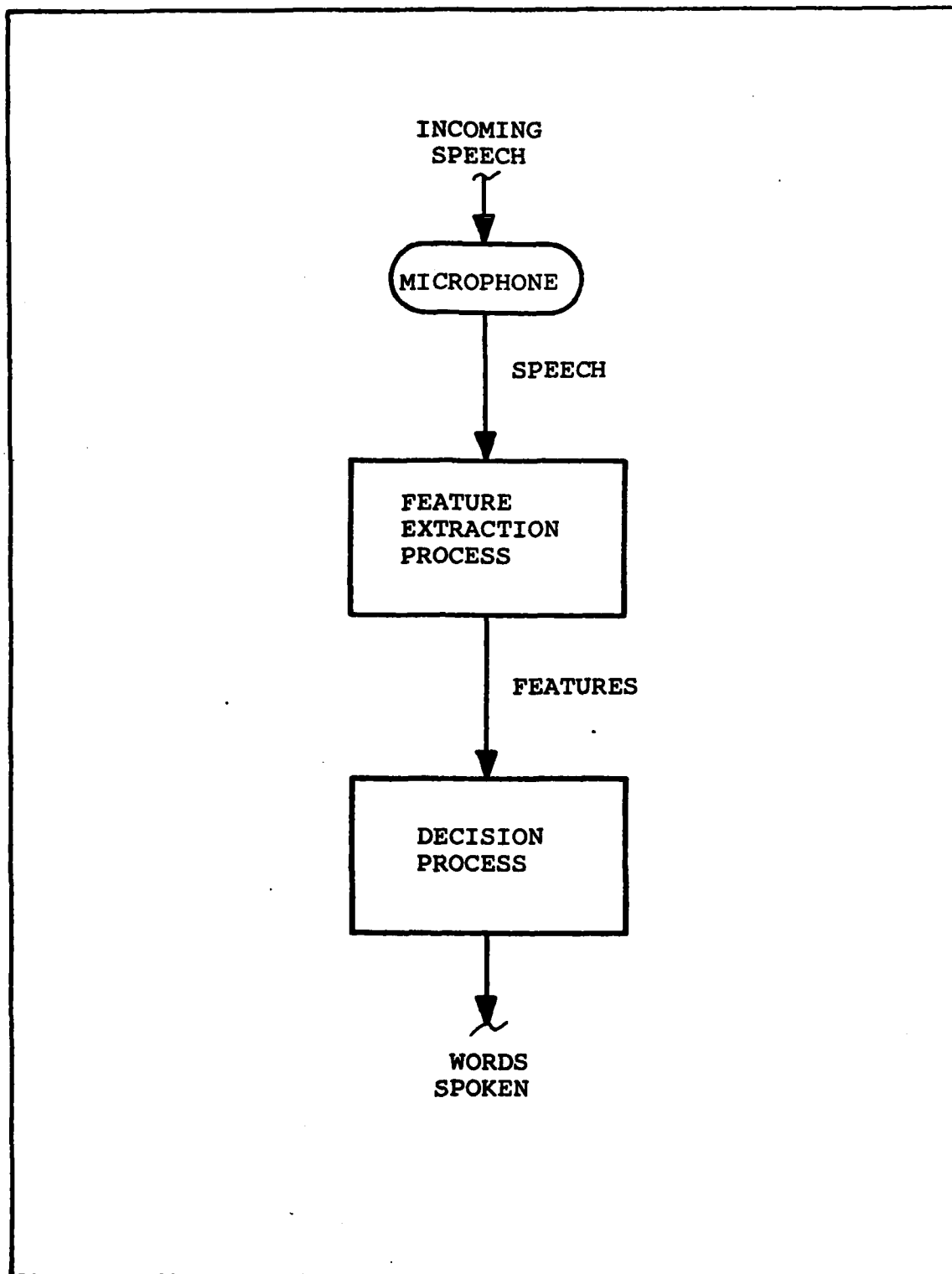


Fig. 1. Block Diagram of Speech Recognition Processes

researchers have attempted to develop feature extraction processes that would perfectly segment speech. However, even if a perfect feature extraction process was developed, there would still remain a number of errors in its output since it is impossible for humans to perfectly reproduce any known features. This fact is easily verified by the inability of humans to repeatably pronounce any word exactly the same, and by the fact that there are vast differences in the speech of different speakers. Therefore, the research effort should not be confined to the development of a perfect feature extraction process, but include as well the development of an efficient decision process that could determine the words spoken given that there will always be a number of errors present in the features extracted from the highly variable input speech.

Problem

The objective of this research was to develop a decision process which would be capable of determining the words spoken from a set of reasonably consistent speech features. Consistent features will be defined as features that appear repeatably, but not necessarily always, for a given word. The more specific objective of this research was to develop an algorithm which would recognize the words from the features extracted by the acoustic processor developed by Karl Seelandt (Ref 15).

The features which Seelandt's algorithm extracts are basic units of speech that will be referred to as phonemes. Although the term "phoneme" classically implies a more exact set of speech segments, this term will be used here to refer to any segment of speech that is defined to be unique.

The output of Seelandt's acoustic processor contains many errors; including substitutions, insertions, and deletions. These errors result from the inadequacies of his algorithm, and by the omission and mutilation of phonemes by speakers. There have been many decision algorithms implemented and suggested in the literature which attempt to overcome problems associated with errors in similar outputs; however, because of the number and types of errors present in Seelandt's output, none of these approaches can be practically implemented. (Refs 1, 7, and 9). In fact, with the possible exception of the algorithm proposed by Kashyap (Ref 9), there are currently no algorithms which are capable of making decisions in a practical manner when there are more than a few errors in the data being examined.

Scope

The recognition algorithm that was developed in this thesis is capable of recognizing isolated words, or phrases that are treated as words. The algorithm is completely independent of the vocabulary in the sense that words or phrases can be added or deleted at any time. However, the accuracy

of the system will be effected by the vocabulary chosen since some words are more likely to be confused with one another than with others. The algorithm can be used as either a speaker dependent or independent system. As with most other recognition systems, the accuracy of the system is higher when using the system as a speaker dependent system since the output of the acoustic analyzer is more consistent for one speaker. The only information that is used in determining the words spoken is obtained from the acoustic output. It should also be noted that no attempt was made to improve Seelandt's algorithm.

General Approach

Because of the high error rate of the acoustic processor, it was necessary to rely on the consistency of the sequences of phonemes, and the errors that typically occur for a word to determine the word spoken. This was accomplished by generating error statistics and phoneme representations for each word in the vocabulary using a set of training speech files. The training files were created by applying the acoustic analyzer to a number of speech samples for each word.

The top five phoneme choices provided by the acoustic processor, and information indicating the accuracy of each choice, for each time segment of speech was implemented in the decision algorithm. Fuzzy set theory was used to combine this information with the error information obtained from

the training files to determine the word spoken. This theory will be presented in Chapter 3.

Because of the number and types of errors present in the acoustic output, it was necessary to develop a transition algorithm capable of producing a word score, indicating the plausibility of a word being the one spoken, regardless of the errors that occurred. Using this algorithm, a score is obtained for each word in the vocabulary, and the word with the highest score is chosen as the word that was spoken.

Sequence of Presentation

The next chapter of this thesis examines the format and accuracy of the acoustic analyzer developed by Seelandt. It is very important that the reader study this chapter carefully to obtain an understanding of the significance of the proposed word recognition algorithm. Chapter 3 presents the basic concepts of fuzzy set theory that are employed in the recognition algorithm that will be described in Chapter 4. Chapters 5 and 6 present possible algorithms for determining a word's phoneme representation and optimizing program parameters, respectively. Preliminary results are given in Chapter 7, and general remarks and conclusions are given in Chapter 8. Chapter 9 provides a detailed list of recommendations. The appendixes contain a recommended approach for designing a continuous speech understanding system, and program documentation.

II. Acoustic Output

The word recognition algorithm developed in this thesis relies solely on the output of the acoustic processor developed by Karl Seelandt (Ref 15). The purpose of this chapter is to examine the format and accuracy of this output.

Basically, the acoustic processor attempts to identify the phonemes that occur in speech. As defined in Chapter 1, a phoneme is a basic unit of speech, combinations of which can be used to form words. The set of phonemes used in this thesis, created from Karl Seelandt's speech, is shown in Table I, and consists of 71 distinct segments of speech that occur in the words "zero" through "nine". However, the acoustic processor, and word recognition algorithm to be presented, can be used with any predefined set of phonemes.

Output Format

Figure 2 illustrates the acoustic output for the word "zero" spoken by GJM. The first column of the output provides the vector number assigned to each segment of speech. The reason that the first vector number is not one is that the noise occurring before and after the actual speech has been removed. The next five columns of the output provide the top five phoneme choices, and their respective weights for each segment. The phoneme chosen is given by the first number of each pair, and corresponds to the phoneme numbers

TABLE I

Set of Phonemes Extracted From Seelandt's Speech

<u>Sound Symbol</u>	<u>From Word</u>	<u>Phoneme Number</u>
XX	noise	1
Z1	zero	2
Z2	zero	3
Z3	zero	4
ER1	zero	5
ER2	zero	6
RA	zero	7
UH	zero	8
OU1	zero	9
OH1	zero	10
OH2	zero	11
OU2	one	12
WU1	one	13
WU2	one	14
AW	one	15
OY1	one	16
OY2	one	17
NX1	one	18
T1	two	19
T2	two	20
T3	two	21
OU3	two	22
OU4	two	23
UA	two	24
TH	three	25
RX1	three	26
RE1	three	27
RE2	three	28

TABLE I (cont'd)

<u>Sound Symbol</u>	<u>From Word</u>	<u>Phoneme Number</u>
EE1	three	29
EE2	three	30
FX1	four	31
FX2	four	32
OH3	four	33
OR	four	34
RX2	four	35
FX3	five	36
FX4	five	37
WA	five	38
AH1	five	39
AH2	five	40
AY1	five	41
AY2	five	42
VX1	five	43
SX1	six	44
SX2	six	45
SX3	six	46
IY1	six	47
IY2	six	48
KX1	six	49
KX2	six	50
SX4	seven	51
EH1	seven	52
EH2	seven	53
VX2	seven	54
EH3	seven	55
NX2	seven	56

TABLE I (cont'd)

<u>Sound Symbol</u>	<u>From Word</u>	<u>Phoneme Number</u>
AE1	eight	57
AE2	eight	58
AE3	eight	59
EE3	eight	60
T4	eight	61
T5	eight	62
NX3	nine	63
AH3	nine	64
AH4	nine	65
AH5	nine	66
AH6	nine	67
IE1	nine	68
IE2	nine	69
NX4	nine	70
XX	noise	71

ZERO

GJM

THE DATE IS-- 8 20 1982
THE TIME IS-- 20 19 24

VECTOR NUMBER *****	FIRST CHOICE *****	SECOND CHOICE *****	THIRD CHOICE *****	FOURTH CHOICE *****	FIFTH CHOICE *****	SCALE FACTOR *****
95	21 100	63 95	56 94	70 93	43 88	.91311530
96	21 100	43 89	63 88	70 87	32 84	.79304210
97	21 100	32 94	70 93	62 92	56 90	.86398080
98	70 100	21 99	62 96	56 93	43 91	.90292700
99	70 100	56 92	21 91	43 88	63 87	.85189500
100	70 100	56 91	63 88	62 87	21 83	.81483860
101	70 100	63 87	12 82	56 82	62 81	.77515350
102	70 100	63 85	12 82	62 80	21 77	.76327380
103	70 100	12 81	63 81	23 78	24 78	.75937090
104	70 100	12 81	13 80	23 80	24 80	.78226070
105	70 100	13 81	24 78	12 77	21 76	.79373070
106	70 100	13 80	24 75	21 74	28 74	.83285790
107	70 100	28 78	13 76	43 74	63 73	.87437260
108	70 100	28 82	43 75	13 72	21 72	.88921780
109	70 100	28 81	43 78	63 73	21 69	.89245720
110	70 100	28 77	43 75	63 75	13 66	.86176660
111	70 100	63 76	43 74	54 71	13 70	.80548720
112	70 100	63 80	54 77	24 76	43 76	.75072310
113	70 100	63 84	12 81	54 81	24 80	.68498410
114	70 100	54 86	12 85	24 83	23 81	.62736520
115	70 100	12 91	54 88	23 86	62 86	.64883750
116	70 100	12 97	62 93	54 91	23 90	.71659460
117	12 100	70 98	21 92	23 92	62 92	.77849600
118	12 100	23 92	21 90	70 89	63 85	.85203830
119	12 100	23 94	33 93	11 92	34 90	.94516810
120	34 100	33 98	23 96	12 95	13 94	1.00000000
121	34 100	08 92	33 92	07 90	13 90	.96905810
122	34 100	08 92	07 87	10 87	11 87	.88132900
123	34 100	08 95	23 94	35 92	09 90	.85554160
124	34 100	23 98	08 97	09 94	11 93	.84072410
125	34 100	12 99	54 99	23 97	08 96	.84773580
126	12 100	54 97	07 91	11 91	34 91	.75137910
127	12 100	07 94	11 93	09 91	13 91	.75798860
128	07 100	09 99	12 98	08 97	11 92	.80022970
129	07 100	09 99	08 97	12 94	33 88	.79255200
130	08 100	09 98	07 97	10 91	11 91	.79841270
131	09 100	08 98	10 98	11 95	33 95	.74377430
132	10 100	33 99	09 98	11 97	08 96	.70680830
133	33 100	11 97	10 95	09 94	34 93	.67387350
134	33 100	11 97	12 93	10 90	34 88	.64743520
135	33 100	11 96	12 92	10 89	07 87	.60208030
136	33 100	11 96	07 94	10 93	12 93	.67539140
137	10 100	11 99	33 98	34 96	07 94	.72672500
138	11 100	10 98	34 95	33 94	08 85	.63285890
139	34 100	10 99	11 95	33 93	08 86	.67046310
140	34 100	11 97	10 95	33 90	12 86	.74917250
141	11 100	10 99	34 98	12 93	33 90	.82464240
142	62 100	11 97	12 95	10 93	71 93	.87923310

Fig. 2. Acoustic Output for Word "Zero"
Spoken by GJM

shown in Table I. The weights assigned to each phoneme have been scaled from 0 to 100 for each vector, with 100 being assigned to the best choice. The last column indicates the scale factor assigned to each vector, and provides a measure of the accuracy of one vector compared to the accuracy of the other vectors in the speech file. This factor has values between 0 and 1, with 1 corresponding to the least accurate vector.

The weights and scale factors given in the acoustic output are derived based on the distances between each prototype phoneme and each segment of speech. The set of distances, D_i , computed for each vector i , can be represented by the set

$$D_i = \{d_{1,i}, d_{2,i}, \dots, d_{c,i}, \dots, d_{n,i}\} \quad (1)$$

with $d_{c,i}$ the distance between the c th prototype phoneme and vector i , and n the number of prototypes. The equations used to compute the weights for the phoneme choice, $X_{c,i}$, and the scale factor, SF_i , for vector i are:

$$X_{c,i} = 100 \times ((\text{Max}\{D_i\} - d_{c,i}) / (\text{Max}\{D_i\} - \text{Min}\{D_i\})) \quad (2)$$

$$SF_i = (\text{Min}\{D_i\}) / \text{Max}\{\text{Min}\{D_1\}, \text{Min}\{D_2\}, \text{Min}\{D_3\}, \dots, \text{Min}\{D_t\}\} \quad (3)$$

where t is the number of vectors in the speech file, and $X_{c,i}$ is the weight assigned to phoneme P_c . Note that

$$\text{Max}\{D_i\} = \text{Max}\{d_{1,i}, d_{2,i}, \dots, d_{c,i}, \dots, d_{n,i}\} \quad (4)$$

$$\text{Min}\{D_i\} = \text{Min}\{d_{1,i}, d_{2,i}, \dots, d_{c,i}, \dots, d_{n,i}\} \quad (5)$$

The logic for scaling the phoneme distances between 0 and 100, and for providing the scale factor for each vector, rather than using the actual distances, is to make the weights for the phoneme choices for a given vector independent of the weights assigned to the phonemes chosen for other vectors. By making these weights independent for each vector, the importance of the value of the minimum distance for each vector can be varied. The manner in which this is done will be discussed in the description of the word recognition algorithm.

Output Accuracy

To provide an understanding of the proposed word recognition algorithm, and the significance of this algorithm, it is necessary to examine the accuracy of the acoustic output. Referring to the phoneme representation for word "zero" given in Table I (phonemes 2 through 11), and to the sample outputs for the word "zero" spoken by five different speakers, shown in Figures 3 through 7; it is easily seen that the accuracy of the acoustic processor is extremely poor, and that often the correct phonemes are not even in the output. In fact, the accuracy is such that any meaningful error percentage would be virtually impossible to derive. To overcome this

ZERO

DCD

THE DATE IS-- 5 25 1982
THE TIME IS-- 19 1 56

VECTOR NUMBER *****	FIRST CHOICE *****	SECOND CHOICE *****	THIRD CHOICE *****	FOURTH CHOICE *****	FIFTH CHOICE *****	SCALE FACTOR *****					
90	62	100	71	93	21	89	20	85	01	81	.81760670
91	02	100	21	95	62	92	20	89	03	87	.89882920
92	02	100	03	84	46	83	20	75	31	74	.81740700
93	19	100	02	99	03	97	20	94	44	90	.94243760
94	20	100	19	89	61	85	02	84	31	82	.82714840
95	20	100	03	87	44	86	46	84	02	83	.90076300
96	20	100	61	85	02	83	03	81	30	81	.84040860
97	20	100	02	99	30	93	46	93	61	93	.96326630
98	20	100	46	98	02	97	31	97	21	96	.97433140
99	29	100	30	97	59	96	20	94	46	94	1.00000000
100	30	100	20	93	46	92	21	89	25	87	.96126880
101	30	100	21	90	60	90	46	88	70	87	.99318220
102	30	100	29	90	46	88	70	88	21	86	.97390340
103	29	100	30	95	46	95	69	91	60	90	.98903580
104	46	100	04	96	69	96	28	95	29	95	.97576470
105	69	100	04	98	22	97	46	93	28	92	.90866070
106	22	100	69	98	70	94	04	92	32	90	.80807590
107	70	100	22	99	69	93	32	91	04	86	.75795470
108	70	100	48	97	22	96	69	87	23	83	.75848590
109	70	100	23	89	63	87	48	84	22	82	.75575530
110	70	100	23	89	63	87	48	82	12	78	.78379240
111	70	100	27	90	23	88	63	85	06	84	.80849080
112	70	100	27	97	23	90	63	88	06	87	.81929250
113	70	100	27	97	23	94	63	94	34	84	.79022160
114	70	100	23	91	63	90	27	87	12	83	.72270700
115	70	100	23	96	63	90	07	88	12	88	.73743130
116	07	100	70	99	23	98	12	93	63	90	.79020180
117	07	100	12	96	23	96	70	96	63	89	.84055130
118	23	100	07	98	12	98	34	96	05	94	.89116410
119	34	100	23	98	12	95	05	93	07	91	.85788100
120	12	100	23	99	52	97	34	93	05	92	.89522270
121	52	100	55	98	23	96	12	95	08	93	.88687070
122	52	100	23	98	12	97	08	96	07	95	.88056220
123	23	100	07	97	12	96	52	94	08	92	.84726150
124	23	100	12	96	07	90	08	90	09	90	.84974180
125	23	100	08	97	09	94	07	91	12	90	.83060780
126	08	100	09	97	23	94	07	91	12	87	.76563950
127	23	100	08	96	07	93	12	91	34	90	.75337810
128	23	100	12	95	54	93	08	89	34	89	.79251320
129	08	100	35	98	38	95	07	94	09	94	.75838940
130	08	100	07	95	09	94	34	90	06	89	.68710600
131	12	100	07	99	06	97	33	97	24	96	.74905610
132	12	100	11	99	33	95	24	92	34	92	.77477740
133	11	100	33	98	10	97	12	96	24	96	.78694440
134	11	100	12	99	10	97	33	97	24	92	.76734290
135	12	100	33	90	11	85	24	85	62	84	.66368070
136	12	100	71	93	62	90	33	88	11	87	.71807770
137	62	100	12	97	71	88	33	87	11	86	.69576100
138	62	100	12	94	71	90	56	84	70	82	.59088270
139	62	100	71	97	12	94	21	86	01	83	.64932520

Fig. 3. Acoustic Output for Word "Zero"
Spoken by DCD

ZERO

JEF

THE DATE IS-- 8 8 1982
THE TIME IS-- 21 37 15

VECTOR NUMBER *****	FIRST CHOICE *****	SECOND CHOICE *****	THIRD CHOICE *****	FOURTH CHOICE *****	FIFTH CHOICE *****	SCALE FACTOR *****
72	18 100	21 93	20 92	56 91	30 88	.99315840
73	18 100	21 95	51 94	31 92	46 90	.94995800
74	18 100	04 83	21 83	45 83	31 76	.90897520
75	18 100	04 88	21 77	45 76	46 72	.89692370
76	04 100	18 96	28 76	70 76	21 75	.96535710
77	04 100	28 79	18 76	47 75	22 73	.87776420
78	04 100	28 83	47 80	22 77	48 74	.83628860
79	04 100	22 85	28 85	47 84	42 82	.77320070
80	04 100	22 92	42 88	47 86	48 83	.72090560
81	22 100	04 98	48 93	42 92	47 92	.75475380
82	05 100	48 98	22 96	04 95	42 94	.74818970
83	05 100	48 97	04 91	42 90	47 90	.69529970
84	05 100	48 94	42 87	47 87	04 86	.65117810
85	05 100	48 90	47 88	42 85	04 83	.66434840
86	05 100	47 88	48 86	42 84	27 81	.69505770
87	05 100	47 86	27 82	48 82	42 80	.71993330
88	05 100	27 85	47 83	48 78	55 78	.74344590
89	05 100	27 90	55 81	47 79	52 77	.79987430
90	05 100	27 92	55 84	53 83	07 80	.85556800
91	05 100	27 97	07 89	55 89	06 88	.91582740
92	27 100	05 92	07 91	35 91	06 90	.88627290
93	27 100	06 91	07 90	35 90	34 87	.84556540
94	27 100	06 92	07 89	35 89	34 87	.82942040
95	27 100	06 92	07 88	34 88	35 88	.81981520
96	27 100	06 91	34 90	35 90	07 88	.82478760
97	27 100	06 92	34 92	35 92	07 89	.84147640
98	27 100	35 98	34 97	06 92	07 91	.86188790
99	35 100	27 94	34 94	08 91	07 90	.81579060
100	35 100	08 93	34 93	39 91	07 90	.74758370
101	35 100	08 96	34 93	39 93	07 91	.69511540
102	35 100	08 98	09 97	07 96	34 96	.70894060
103	07 100	34 98	35 97	08 95	09 94	.67848210
104	07 100	08 94	09 93	34 92	11 91	.64661180
105	07 100	34 92	08 91	09 91	10 90	.73583630
106	07 100	08 98	09 98	10 92	38 90	.77581350
107	09 100	08 98	07 93	38 91	10 84	.74659140
108	08 100	09 95	38 91	07 85	11 85	.72431750
109	08 100	09 92	38 92	07 88	10 88	.72031510
110	08 100	09 99	34 96	35 95	10 91	.68119030
111	09 100	08 99	07 96	34 93	35 93	.64286700
112	07 100	08 99	09 97	34 94	38 92	.67485710
113	07 100	08 99	38 93	34 91	09 90	.68423820
114	08 100	07 92	38 92	11 91	34 91	.67403800
115	08 100	34 95	38 93	09 91	12 91	.70401700
116	08 100	12 99	09 96	38 96	10 95	.81704260

Fig. 4. Acoustic Output for Word "Zero"
Spoken by JEF

ZERO

LMR

THE DATE IS-- 7 9 1982
THE TIME IS-- 10 35 12

VECTOR NUMBER *****	FIRST CHOICE *****	SECOND CHOICE *****	THIRD CHOICE *****	FOURTH CHOICE *****	FIFTH CHOICE *****	SCALE FACTOR *****
101	23 100	70 98	12 95	33 92	24 90	.95155790
102	70 100	23 99	12 90	13 87	33 87	.88991430
103	70 100	23 96	13 88	33 86	12 84	.89634330
104	70 100	23 95	33 90	13 88	57 86	.94451220
105	70 100	23 94	28 89	57 89	33 86	.95383370
106	70 100	23 96	28 95	57 89	48 88	.97316340
107	70 100	28 99	23 97	48 95	47 91	1.00000000
108	48 100	28 98	47 97	70 97	23 95	.99166350
109	47 100	48 98	27 91	05 88	70 88	.89924330
110	47 100	48 91	27 90	42 88	05 87	.82710270
111	47 100	05 98	27 96	42 94	48 94	.87139420
112	05 100	27 96	47 91	48 91	42 87	.80782310
113	27 100	05 98	47 91	48 91	42 86	.83385000
114	05 100	27 99	47 90	48 89	07 87	.81527980
115	05 100	27 97	07 87	35 87	47 87	.80135390
116	05 100	27 96	41 94	35 91	07 89	.82540080
117	05 100	41 98	35 97	27 94	07 92	.85626520
118	05 100	35 95	07 92	41 91	27 87	.83018060
119	05 100	07 98	35 97	41 97	52 93	.91199110
120	52 100	41 99	35 97	07 96	05 94	.91404380
121	52 100	35 98	07 94	05 93	41 93	.88633510
122	52 100	41 98	07 95	35 95	08 91	.88864350
123	52 100	07 98	35 98	41 98	39 93	.91327940
124	52 100	07 98	41 97	35 95	39 95	.90638920
125	07 100	39 97	41 96	52 96	35 95	.90546430
126	07 100	39 99	41 98	10 97	52 95	.92884700
127	39 100	07 96	10 95	41 95	52 93	.90774760
128	39 100	41 99	07 97	10 96	52 91	.93706050
129	41 100	39 99	07 95	52 94	35 93	.93934140
130	41 100	39 94	52 92	07 90	10 87	.89274660
131	41 100	52 98	39 97	35 91	07 90	.93388740
132	41 100	52 99	39 93	07 88	35 87	.90551440
133	52 100	41 95	39 92	35 85	07 84	.85531030
134	52 100	41 95	39 89	35 84	07 83	.82273390
135	52 100	41 91	39 82	53 80	07 79	.76898570
136	52 100	41 85	39 83	53 83	07 78	.72995780
137	52 100	41 83	53 83	39 80	07 76	.68760880
138	52 100	53 88	41 83	39 80	07 76	.70787350
139	52 100	53 91	39 86	41 82	07 79	.75222250
140	52 100	53 95	39 89	41 84	09 83	.75002690
141	52 100	53 97	39 92	09 90	08 89	.83117820
142	39 100	52 97	08 96	07 95	53 95	.89778450
143	39 100	07 96	08 96	09 95	35 95	.83792300
144	39 100	07 99	08 99	09 98	33 95	.89643600
145	07 100	33 97	35 97	26 92	39 92	.85804230
146	07 100	35 97	33 94	09 91	11 91	.80053440
147	07 100	12 95	33 94	09 93	08 92	.85925790
148	07 100	33 99	12 98	11 93	26 90	.88728750
149	33 100	01 97	12 97	62 97	07 96	.90737920

Fig. 5. Acoustic Output for Word "Zero"
Spoken by LMR

ZERO

MJK

THE DATE IS-- 5 20 1982
THE TIME IS-- 2 54 39

VECTOR NUMBER *****	FIRST CHOICE *****	SECOND CHOICE *****	THIRD CHOICE *****	FOURTH CHOICE *****	FIFTH CHOICE *****	SCALE FACTOR *****
98	03 100	01 93	71 84	02 82	62 66	.82128940
99	02 100	01 89	03 84	71 79	20 72	.81486480
100	01 100	02 75	19 71	03 70	71 69	.69117470
101	01 100	19 81	03 75	71 67	02 64	.64976470
102	01 100	03 74	19 71	71 68	02 52	.65773640
103	01 100	03 88	71 66	02 65	19 65	.63326690
104	03 100	01 97	02 82	19 79	71 65	.78161590
105	03 100	19 93	02 91	01 83	20 74	.83109990
106	03 100	19 99	20 75	02 73	29 65	.73087350
107	03 100	19 89	20 81	29 66	59 64	.71808520
108	03 100	20 93	19 75	29 71	59 66	.78974500
109	20 100	03 89	44 81	30 80	61 79	.91133280
110	30 100	46 99	21 93	45 92	20 91	.99011790
111	70 100	04 99	30 99	45 98	21 95	1.00000000
112	04 100	70 97	28 91	22 87	30 83	.94437750
113	04 100	70 98	28 91	22 87	13 81	.93398200
114	70 100	04 94	28 91	22 89	48 87	.86283570
115	70 100	48 95	28 93	22 89	04 87	.78195260
116	70 100	48 97	22 93	28 90	23 84	.74313770
117	70 100	48 88	22 87	23 85	28 81	.71269390
118	70 100	27 89	48 89	23 88	63 82	.73021480
119	70 100	27 94	23 94	06 89	48 87	.77017440
120	27 100	23 94	70 93	07 92	06 89	.75030090
121	27 100	23 94	07 92	70 90	06 88	.71738240
122	27 100	23 95	70 91	07 89	33 88	.68471430
123	23 100	27 95	33 91	70 90	07 89	.67089920
124	23 100	33 92	07 88	26 87	27 87	.63038190
125	23 100	33 96	07 92	26 89	06 88	.63559440
126	33 100	23 96	07 95	12 90	34 90	.64036700
127	33 100	07 99	34 92	23 91	06 89	.64739980
128	07 100	33 95	34 92	27 91	06 87	.63049130
129	07 100	34 94	27 93	33 93	35 88	.65319100
130	07 100	34 96	27 94	33 93	35 93	.69736360
131	07 100	34 95	33 93	35 93	08 92	.71064220
132	07 100	08 98	09 98	34 96	10 93	.72598510
133	08 100	09 99	07 97	10 93	34 92	.69765400
134	09 100	08 98	07 96	10 93	11 91	.69873360
135	09 100	08 99	10 97	07 95	11 95	.73280530
136	10 100	08 99	11 98	09 97	33 97	.73978330
137	12 100	08 98	33 98	11 97	09 95	.70654710
138	12 100	11 96	33 94	13 89	09 87	.61810920
139	12 100	11 92	33 91	13 89	09 86	.58621560
140	12 100	11 92	33 92	08 91	09 91	.65384550
141	12 100	11 92	07 91	23 91	33 91	.66156430
142	12 100	11 90	23 90	33 90	10 88	.64918850
143	12 100	23 98	11 96	10 94	33 93	.68718060
144	23 100	12 97	11 94	24 91	07 88	.69230060
145	12 100	11 94	23 92	33 92	34 88	.68498370
146	12 100	62 97	33 96	11 95	01 90	.74130050

Fig. 6. Acoustic Output for Word "Zero"
Spoken by MJK

ZERO

RLC

THE DATE IS-- 5 29 1982
THE TIME IS-- 14 19 23

VECTOR NUMBER *****	FIRST CHOICE *****	SECOND CHOICE *****	THIRD CHOICE *****	FOURTH CHOICE *****	FIFTH CHOICE *****	SCALE FACTOR *****
80	03 100	19 92	02 89	01 83	20 68	.83642510
81	03 100	19 95	20 83	02 78	01 67	.87908170
82	03 100	20 92	19 83	44 76	29 74	.96659320
83	20 100	61 92	25 88	51 84	46 78	.98333780
84	32 100	20 97	21 90	18 86	46 85	.96238590
85	70 100	21 93	63 90	04 87	18 87	.85514750
86	70 100	21 94	04 89	23 89	18 88	.87223290
87	21 100	70 93	28 92	04 86	18 86	.90579780
88	28 100	22 97	60 93	21 91	70 91	.96419890
89	60 100	22 99	30 92	70 90	18 89	.95117830
90	22 100	60 95	30 94	28 91	69 88	.93124870
91	69 100	22 99	30 94	28 93	70 93	1.00000000
92	70 100	28 97	69 91	22 87	48 86	.94499770
93	70 100	48 86	04 84	63 84	28 83	.82626210
94	70 100	63 86	48 82	04 79	22 77	.79448350
95	70 100	63 92	48 91	43 80	04 79	.79253020
96	70 100	63 96	48 92	05 85	23 85	.81578730
97	63 100	70 99	23 98	54 94	26 92	.84678410
98	63 100	23 99	26 97	70 97	27 94	.80866450
99	26 100	23 99	06 96	70 96	54 94	.77236000
100	26 100	07 97	54 97	23 96	06 95	.73679290
101	07 100	06 98	26 98	27 96	23 94	.72435360
102	06 100	26 95	07 94	27 94	35 88	.69644150
103	27 100	06 98	35 98	26 95	07 90	.73300430
104	35 100	27 95	06 91	07 86	26 86	.67980830
105	35 100	27 97	07 95	06 93	34 88	.74359270
106	07 100	35 96	27 92	55 92	06 91	.73468580
107	07 100	35 92	27 89	26 87	06 86	.71449790
108	07 100	35 96	27 91	34 90	26 88	.75898530
109	35 100	34 97	07 96	27 90	06 88	.75825920
110	35 100	34 96	08 91	07 90	39 90	.71297000
111	35 100	34 99	07 95	08 95	39 95	.79819630
112	07 100	35 94	39 94	08 93	54 93	.76658480
113	07 100	08 92	23 92	26 92	09 90	.72521780
114	07 100	08 97	23 93	09 91	34 91	.75838840
115	08 100	34 97	07 96	23 94	12 93	.77299700
116	08 100	12 99	34 97	11 94	09 93	.75455530
117	12 100	11 89	34 89	08 88	09 85	.63196660
118	12 100	54 86	11 82	24 80	33 80	.54840840
119	12 100	54 83	11 79	24 78	33 78	.46845190
120	12 100	62 86	54 81	70 79	11 78	.47375940
121	12 100	62 93	70 84	21 83	54 82	.53143450
122	12 100	62 98	70 87	21 84	54 80	.58056580
123	62 100	12 96	21 89	01 84	71 84	.63793120
124	62 100	01 99	71 90	12 86	21 76	.63102670

Fig. 7. Acoustic Output for Word "Zero"
Spoken by RLC

error rate it was necessary to rely on the consistency of the errors that are made, and the sequences of phonemes that are output for each word. Obviously, this consistency does not exist for the first phoneme choices for a word. Because of this fact, more choices were examined, and it was found that although not obvious, consistencies do appear to exist in the top choices of the output. Although 5 choices were used in this thesis, the word recognition algorithm can easily be altered to consider any number of choices.

At present, there are no algorithms known to the author which can account for the number of errors present in this output, or that can identify the non-obvious consistencies that appear to exist. Also, there are no known algorithms currently available which can efficiently use the information provided in a number of alternate choices for each segment of speech. Therefore it was necessary to develop a unique algorithm which could:

1. account for a large number of errors in the output;
2. determine the consistencies that occur in the output to recognize the word spoken; and
3. efficiently use the information provided by a number of alternative choices for a given vector.

These objectives were accomplished using fuzzy set theory, which will be presented in the following chapter.

III. Fuzzy Set Theory

Introduction

The concept of Fuzzy Set Theory was first introduced by Lofti A. Zadeh in 1965 (Ref 19). Since then numerous articles have been published concerning the algebraic properties, and possible applications of this theory. However, the successes of fuzzy set theory have been extremely limited. Perhaps the major reason that accounts for the limited success of this relatively new theory, is that the applications for which it appears to be suited deal with problems where there have only been limited successes. For example the areas of pattern recognition and Artificial Intelligence appear to be ideal fields for the application of this theory, and at present there are no methods which appear to be superior to any other in either of these areas.

Fuzzy Set Theory

Fuzzy set theory, as the name implies, provides a theory for sets in which the transition from membership to non-membership is not clearly defined (fuzzy) (Ref 20). For example, the set of people who are young is not clearly defined, and the transition from membership to non-membership can not be clearly established. To account for this fuzziness, the members of a fuzzy set are assigned grades of membership in the interval from zero to one. A fuzzy set, S , will then be of

the form:

$$S = \{x_1/u(x_1), x_2/u(x_2), \dots, x_i/u(x_i), \dots, x_n/u(x_n)\} \quad (6)$$

where x_i is an element of set S with a grade of membership of $u(x_i)$ in the set. Continuing the example of the set consisting of young people, one possible fuzzy set could be

$$S(\text{young}) = \{10/1.0, 25/0.9, 35/0.8, 50/0.5, 75/0.01\} \quad (7)$$

where the ages, or elements, of the set are 10, 25, 35, 50, and 75; and their respective grades of membership are 1.0, 0.9, 0.8, 0.5, and 0.01.

The grades of membership can be assigned either subjectively, as the ones shown above, or may be assigned by using a mathematical equation. An example of an intuitive equation that could be used to determine the grades of membership of the elements in the set of young people, taken from Ref 17, is shown below.

$$\begin{aligned} u(x_i) &= \{1, \text{ for } x_i \leq 25, \\ &= \{(1 + ((x_i - 25)/5)^2)^{-1}, \quad \text{for } x_i > 25 \end{aligned} \quad (8)$$

where $u(x_i)$ is the grade of membership of the element x_i in the set $S(\text{young})$.

Algebraic Properties

From the discussion presented thus far, it may appear

that fuzzy set theory is an extremely informal mathematical tool; however, since the introduction of this theory, there have been numerous papers written which have attempted to provide a more formal theory. Some of the formal algebraic properties that have been suggested include:

Equality: Two fuzzy sets, A and B, are equal if and only if $u_A(x) = u_B(x)$ for all x, where x is an element of the set of all elements.

Containment: A fuzzy set A is contained in, or is a subset of a fuzzy set B, written $A < B$, if and only if $u_A(x) \leq u_B(x)$ for all x.

Complementation: A' is the complement of the fuzzy set A if and only if $u_{A'}(x) = 1 - u_A(x)$, for all x.

Intersection: The intersection of the fuzzy sets A and B, denoted by $A \wedge B$, is given by

$$u_{A \wedge B}(x) = \text{Min } (u_A(x), u_B(x)) \text{ for all } x \quad (9)$$

and is defined as the largest fuzzy set contained in both A and B.

Union: The union of the fuzzy sets A and B, denoted $A \cup B$, is given by

$$u_{A \cup B}(x) = \text{Max } (u_A(x), u_B(x)) \text{ for all } x \quad (10)$$

and is defined as the smallest fuzzy set containing both A and B.

An extensive examination of these properties and others that have been proposed, can be found in the references (Refs 8 and 19).

Although formalization is generally useful, strict adherence to the formal theory that some have proposed, may significantly diminish its advantages and possible applications. For instance, the operators "Min" and "Max" are used extensively for the sake of formality. Although these operators are definitely of value, the author shares the view of Dubois and Prade who state

"... the choice of an operator is always a matter of context, (and) mainly depends upon the real world situation which is to be modeled. In other words, all mathematical properties regarding the class of fuzzy set theoretic operators must be interpreted at an intuitive level." (Ref 2).

Therefore, any operator which appears reasonable, such as the "average" or "product" operators defined below, should also be considered when developing a fuzzy algorithm (i.e. an algorithm employing fuzzy set theory).

Average: The average of two fuzzy sets A and B, to produce fuzzy set C, is given by

$$u_C(x) = (u_A(x) + u_B(x))/2 \text{ for all } x \quad (11)$$

Product: The product of two fuzzy sets A and B, to produce fuzzy set C, is given by

$$u_C(x) = u_A(x) \times u_B(x) \text{ for all } x \quad (12)$$

Decision Making Using Fuzzy Sets

An important application of fuzzy set theory is in making decisions when a number of factors must be considered. Ronald R. Yager has written two excellent articles describing the application of fuzzy set theory to multiple objective decision-making (Refs 17 and 18). The decision scheme that follows is similar to the one proposed by Yager, with a few notable exceptions.

Perhaps the best way to present the proposed decision scheme is by means of an example. Assume that a businessman must decide what job to take given a choice of 4 jobs located in the following cities:

- A. Anchorage, Alaska;
- B. Boston, Massachusetts;
- C. Chicago, Illinois;
- D. Denver, Colorado.

He decides that he will base his decision on the following three factors:

- F1: salary vs cost of living;
- F2: advancement opportunities; and
- F3: his wife's preference.

After researching the cost of living in each city, and the possible employers, and after numerous "discussions" with his wife, he subjectively arrives at the following fuzzy sets relating each job to each of the three factors.

$$F_1 = \{A/0.4, B/0.5, C/0.6, D/0.9\} \quad (13)$$

$$F_2 = \{A/0.8, B/0.9, C/0.7, D/0.5\} \quad (14)$$

$$F_3 = \{A/0.01, B/0.3, C/0.6, D/0.7\} \quad (15)$$

If he assumes that all three factors are equally important, then one method he could employ to arrive at a decision would be to choose the job provided by the maximum of the intersection of the three sets. Evaluating the intersection, which was defined previously, gives

$$D = F_1 \wedge F_2 \wedge F_3 = \{A/0.01, B/0.3, C/0.6, D/0.5\} \quad (16)$$

therefore the job located in Chicago would be the best choice.

There are basically two problems with this approach. First, in all likelihood, the importance of each factor relative to the other factors, is different; and second, this method essentially only considers the worst factor for each of the four jobs. Before continuing the example, a discussion of how these deficiencies can be resolved will be given.

The first inadequacy can be eliminated by applying an equation to each fuzzy set being considered. For example, each grade of membership in a set can be divided by a constant and raised to power as shown below for an arbitrary fuzzy set A.

$$(A/e)^f = \{x_1/[(u(x_1)/e)^f], x_2/[(u(x_2)/e)^f], \dots, x_n/[(u(x_n)/e)^f]\} \quad (17)$$

where e and f will be referred to as fuzzy variables, and the expression $(A/e)**f$ will be referred to as a fuzzy equation. By varying the values of the fuzzy variables for different fuzzy sets involved in a decision, the designer can vary the importance of each factor on the final decision.

The second problem discussed above can be resolved by employing an operation which considers all the factors in the final decision, such as the average or product operations presented earlier. Therefore, by using an operation such as the average operation, and by using fuzzy equations the importance of each factor on the final decision can be adjusted, and all the factors can be considered.

It should be noted that any number of operations other than the average or product operations could be used, and a variety of fuzzy equations could be used that would also overcome the deficiencies discussed above. An interesting example of possible fuzzy equations that could be implemented (which is similar to contrast intensification discussed in reference 21) would be any equation that would use one set of fuzzy variables when the grade of membership is above a specified threshold, and would use another set of variables when the grade of membership is below the threshold. As an example of how such an equation would be valuable, consider the factor F_1 of the above example. If the salary offered for one of the jobs is below some level, then obviously this factor should be weighted heavily in the final decision; however, if

all the salaries offered are within a reasonable range then the importance of this factor may become negligible.

One question that has not been addressed is how the fuzzy variables should be determined to produce the best decision. This of course depends on the decision being made, and the variables may be chosen arbitrarily or by some appropriate algorithm. One approach would be to derive the variables based on the performance of the decision scheme for similar situations where the results are known. In the job example, the businessman could survey colleagues who were faced with similar decisions for choosing a job, and ask them how satisfied they feel they would have been had they taken the other jobs. Using this information a set of variables could be derived based on the set that would have produced the most satisfaction for his colleagues.

Continuing the previous example, assume the businessman decides to use the average operator and the fuzzy equation discussed earlier. After generating a reasonable set of fuzzy variables, and applying the fuzzy equation to each factor, the fuzzy set for each factor become

$$(F1/1)^{0.5} = \{A/0.63, B/0.71, C/0.77, D/0.95\} \quad (18)$$

$$(F2/2)^{2.0} = \{A/0.16, B/0.20, C/0.12, D/0.06\} \quad (19)$$

$$(F3/1)^{0.4} = \{A/0.06, B/0.62, C/0.82, D/0.87\} \quad (20)$$

Then applying the average operator gives

$$(F_1 + F_2 + F_3)/3 = \{A/0.28, B/0.51, C/0.57, D/0.63\} \quad (21)$$

and therefore the job located in Denver would be the best choice.

General Remarks

In this chapter the basic concepts of fuzzy set theory were presented, and a simple decision scheme was proposed for making decisions when a number of factors must be considered. As demonstrated in the remainder of this thesis, this theory can provide a basis for formalizing an otherwise heuristic approach to a problem. Thus, the formal properties of fuzzy sets suggested in the literature, should only be adhered to when there is at least an intuitive motive for doing so. Because of the ability of this theory to account for the imprecisions that are involved in decision making, this theory may provide an ideal theory applicable to many Pattern Recognition and Artificial Intelligence problems that currently have no solution.

IV. Word Recognition Algorithm

Introduction

This chapter presents the decision algorithm used to determine the word spoken given the acoustic output discussed in Chapter 2. The algorithm relies solely on the information provided by the acoustic processor, and on fuzzy statistics derived from applying the recognition algorithm to a set of training speech files, to produce a word score indicating the plausibility of a word being the one actually spoken. Because of the error-prone and variable nature of the acoustic output, it was necessary to develop an algorithm which would produce a word score regardless of the number or type of errors made. Although it may be possible to limit the number of words considered, the decision scheme currently computes a score for every word in the vocabulary to determine the word spoken. A possible approach for limiting the number of words considered will be suggested in the recommendations.

The word recognition algorithm can basically be divided into the application of three sub-algorithms.

1. A vector scoring algorithm which generates a score indicating the plausibility of a word phoneme occurring for a given vector of the acoustic output.

2. A transition algorithm which determines the word phoneme that was most likely uttered for each vector of

speech, along with the most likely error that occurred.

3. An algorithm which determines the total word score based on the results of the previous sub-algorithms, and on any factors not previously considered. Each of these sub-algorithms will be discussed in this chapter.

To derive a meaningful word score, the decision algorithm must take into account a number of factors. This is accomplished using the concepts of fuzzy set theory presented in Chapter 3. This theory provides a method to formally present the algorithm, and provides valuable insight required to improve or expand the recognition scheme.

As mentioned earlier, the algorithm uses fuzzy statistics based on past recognition attempts to compute the word scores. These statistics are generated from the error information, and results obtained from the recognition algorithm. A thorough discussion of fuzzy statistics will be presented in the next section.

Fuzzy Statistics

The name "fuzzy statistics" is derived from the fact that the statistics used by the word recognition algorithm are determined by applying a fuzzy algorithm. This algorithm will be presented at the end of this chapter since it requires information obtained from applying the recognition algorithm. The statistics are collected on a word basis from a set of

training files given for each word. The logic for collecting statistics for each word rather than collecting overall statistics follows:

1. The problem of choosing one universal phoneme representation for a word that would be valid for a variety of speakers, and that would account for slight variations in a word's pronunciation is extremely difficult; but can be somewhat overcome by collecting separate statistics for each word. Also, a set of overall statistics may become distorted from variations peculiar to specific words.

2. There is a possibility that a word's phoneme representation is erroneous, and word based statistics can be used to automatically correct errors in the representation. This is in fact the problem that arose in this thesis. The output of the acoustic processor was so variable that the word phoneme representation that provided the highest accuracy was completely different from the expected representation.

3. Because of memory constraints, it would be impractical to collect overall phoneme statistics based on the surrounding phonemes (for other than a small phoneme set). By collecting statistics on a word basis, this problem is overcome since the position of each word phoneme is fixed. The reason that it is desirable to have statistics for each phoneme based on the surrounding phonemes, is that the surrounding phonemes may have a major effect on the types of

errors that occur for particular phonemes. Although this problem can also be solved by using appropriate phonological rules, these rules may not be always valid, and are difficult to implement.

Figures 8 through 12 illustrate the data that are maintained for each word (actual data are for the word "zero").

Figure 8 provides

- the number of training speech files that were used to determine the fuzzy statistics for the word;
- the number of recognition attempts that were made for the word;
- the number of times the word was correctly recognized;
- the average recognition score;
- the average difference between the word's recognition score and the next highest score (only computed when word is correctly recognized); and
- the minimum difference between the word's recognition score and the next highest score (also only computed when word is correctly recognized).

Except for the number of training files used to generate statistics, none of this information is used in the recognition algorithm; however, this information is used by the routine for optimizing the fuzzy variables, and also serves as an indication of the algorithm's performance.

Figure 9 illustrates the overall and word fuzzy variables

THE STATISTICS FOR ZERO FOLLOW.

THE SPEAKER IS "GJM"

OF TIMES SPOKEN = 5

OF RECOGNITION ATTEMPTS = 6

OF TIMES CORRECTLY RECOGNIZED = 6

AVERAGE RECOGNITION SCORE = 0.987

AVERAGE DIFFERENCE BETWEEN WORD SCORE
AND NEXT HIGHEST SCORE = 0.098

MINIMUM DIFFERENCE BETWEEN WORD SCORE
AND NEXT HIGHEST SCORE = 4.5E-02

Fig. 8. Performance Data Maintained for Each Word (Data Shown is for the Word "Zero")

THE OVERALL FUZZY VARIABLES THAT WERE USED FOLLOW

STHR =	1.0E+00	SUBE =	1.0E+00	SUBF =	5.0E-01
INSE =	1.3E+00	INSF =	5.0E-01		
DELE =	1.0E+00	DELF =	8.0E-01	DELC =	1.0E-01
DCNE =	1.0E+00	DCNF =	1.2E+00	DCNG =	5.0E-01
SFE =	2.0E+00	SFF =	2.0E+00		
CHVE =	4.0E+00	CHVF =	2.5E-01		
STATE=	1.0E+00	STATF=	3.0E+00	STATG=	0.0E+00
THR1E=	1.0E+00	THR1F=	7.5E-01		
THR2E=	1.0E+00	THR2F=	5.0E-01		

THE WORD FUZZY VARIABLES FOLLOW

WSTHR =	8.0E-01	WSUBE =	1.0E+00	WSUBF =	5.0E-01
WINSE =	1.3E+00	WINSF =	5.0E-01		
WDELE =	1.0E+00	WDELF =	8.0E-01	WDELC =	1.0E-01
WDCNE =	1.0E+00	WDCNF =	1.2E+00	WDCNG =	5.0E-01
WSFE =	2.0E+00	WSFF =	2.0E+00		
WCHVE =	4.0E+00	WCHVF =	2.5E-01		
WSTATE=	1.0E+00	WSTATF=	3.0E+00	WSTATG=	7.0E-01
WTHR1E=	1.0E+00	WTHR1F=	7.5E-01		
WTHR2E=	1.0E+00	WTHR2F=	5.0E-01		

Fig. 9. Example of the Overall and Word Fuzzy Variables

TIMES EACH WORD PHONEME IS DELETED

PHONEME REP =	21	70	7	12	62
# DELETED =	4	0	0	1	0

Fig. 10. Deletion Statistics Generated For the Word "Zero"

TIMES PHONEME X SUBSTITUTED FOR WORD PHONEME

X	WORD PHONEMES				
	21	70	7	12	62
1	0.000	0.489	0.000	0.000	1.984
2	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000
4	0.452	1.921	0.000	0.000	0.000
5	0.000	0.514	0.507	0.000	0.000
6	0.000	1.366	2.637	0.000	0.000
7	0.000	1.521	4.057	2.012	0.556
8	0.000	0.000	3.277	2.322	0.000
9	0.000	0.000	2.628	2.092	0.000
10	0.000	0.000	2.095	1.782	0.000
11	0.000	0.493	1.601	1.998	0.475
12	0.459	2.772	1.065	1.487	1.016
13	0.000	1.706	0.994	0.522	0.000
14	0.000	0.000	0.000	0.000	0.000
15	0.000	0.000	0.000	0.000	0.000
16	0.000	0.000	0.000	0.000	0.000
17	0.000	0.000	0.000	0.000	0.000
18	0.469	0.432	0.000	0.000	0.000
19	0.000	0.000	0.000	0.000	0.000
20	0.000	0.000	0.000	0.000	0.000
21	0.841	0.948	0.000	0.000	0.000
22	0.000	0.479	0.000	0.000	0.000
23	0.437	2.830	2.165	0.000	0.000
24	0.000	1.938	0.000	1.150	1.396
25	0.000	1.014	0.000	0.000	0.000
26	0.000	1.931	1.639	0.565	0.499
27	0.000	3.180	3.272	0.000	0.000
28	0.000	1.351	0.000	0.000	0.000
29	0.000	0.000	0.000	0.000	0.000
30	0.000	0.000	0.000	0.000	0.000
31	0.000	0.000	0.000	0.000	0.000
32	0.612	0.000	0.000	0.000	0.000
33	0.000	2.487	3.570	1.480	0.934
34	0.000	0.000	4.106	3.342	2.054
35	0.000	0.000	3.771	0.654	0.000

Fig. 11. Substitution Statistics for Word "Zero" for Prototype Phonemes 1 Through 35

* TIMES PHONEME X SUBSTITUTED FOR WORD PHONEME

X	WORD PHONEMES				
	21	70	7	12	62
36	0.000	0.000	0.000	0.000	0.000
37	0.000	0.000	0.000	0.000	0.000
38	0.000	0.000	2.599	1.422	0.488
39	0.000	0.000	1.456	0.000	0.000
40	0.000	0.000	0.000	0.000	0.000
41	0.000	0.000	0.000	0.000	0.000
42	0.000	0.485	0.000	0.000	0.000
43	0.592	1.005	0.000	0.000	0.000
44	0.000	0.000	0.000	0.000	0.000
45	0.000	0.000	0.000	0.000	0.000
46	0.000	0.000	0.000	0.000	0.000
47	0.000	1.024	0.000	0.000	0.000
48	0.000	1.488	0.000	0.000	0.000
49	0.000	0.000	0.000	0.000	0.000
50	0.000	0.000	0.000	0.000	0.000
51	0.000	0.000	0.000	0.000	0.000
52	0.000	0.000	0.000	0.000	0.000
53	0.000	0.000	0.000	0.000	0.000
54	0.000	1.024	0.548	1.706	1.632
55	0.000	0.000	0.000	0.000	0.000
56	0.763	1.030	0.000	0.000	0.469
57	0.000	0.000	0.000	0.000	0.000
58	0.000	0.000	0.000	0.000	0.000
59	0.000	0.000	0.000	0.000	0.000
60	0.000	0.000	0.000	0.000	0.000
61	0.000	0.443	0.000	0.000	0.000
62	0.439	2.005	0.000	0.615	4.085
63	0.644	2.635	0.000	0.000	0.000
64	0.000	0.000	0.000	0.000	0.000
65	0.000	0.000	0.000	0.000	0.000
66	0.000	0.000	0.000	0.000	0.000
67	0.000	0.000	0.000	0.000	0.000
68	0.000	0.000	0.000	0.000	0.000
69	0.000	0.483	0.000	0.000	0.000
70	0.864	4.448	1.528	0.000	0.000
71	0.000	0.450	0.000	0.000	2.163

Fig. 11 (Cont'd). Substitution Statistics for Word "Zero" for Prototype Phonemes 36 Through 71

* TIMES PHONEME X INSERTED FOR WORD PHONEME

X	WORD PHONEMES				
	21	70	7	12	62
1	0.425	0.000	0.000	0.000	0.000
2	0.000	0.000	0.000	0.000	0.000
3	0.796	0.000	0.000	0.000	0.000
4	0.455	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.542	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	2.153	0.536	0.000
9	0.000	0.000	1.978	0.874	0.000
10	0.000	0.000	0.566	0.532	0.000
11	0.000	0.000	0.589	0.568	0.000
12	0.458	0.000	0.597	0.000	0.000
13	0.000	0.000	0.000	0.000	0.000
14	0.000	0.000	0.000	0.000	0.000
15	0.000	0.000	0.000	0.000	0.000
16	0.000	0.000	0.000	0.000	0.000
17	0.000	0.000	0.000	0.000	0.000
18	0.000	0.000	0.000	0.000	0.000
19	0.000	0.000	0.000	0.000	0.000
20	0.488	0.000	0.000	0.000	0.000
21	0.000	0.000	0.000	0.000	0.000
22	0.000	0.000	0.000	0.000	0.000
23	0.443	0.000	0.574	0.000	0.000
24	0.430	0.000	0.000	0.000	0.000
25	0.000	0.750	0.000	0.000	0.000
26	0.000	0.000	0.000	0.000	0.000
27	0.000	0.000	0.574	0.000	0.000
28	0.000	0.000	0.000	0.000	0.000
29	0.000	0.000	0.000	0.000	0.000
30	0.000	0.000	0.000	0.000	0.000
31	0.474	0.477	0.000	0.000	0.000
32	0.479	0.000	0.000	0.000	0.000
33	0.000	0.000	1.698	0.582	0.000
34	0.000	0.000	1.484	0.000	0.000
35	0.000	0.000	1.387	0.000	0.000

Fig. 12. Insertion Statistics for Word "Zero"
for Prototype Phonemes 1 Through 35

* TIMES PHONEME X INSERTED FOR WORD PHONEME

X	WORD PHONEMES				
	21	70	7	12	62
36	0.000	0.000	0.000	0.000	0.000
37	0.000	0.000	0.000	0.000	0.000
38	0.000	0.000	1.956	0.000	0.000
39	0.000	0.000	0.577	0.000	0.000
40	0.000	0.000	0.000	0.000	0.000
41	0.000	0.000	0.000	0.000	0.000
42	0.000	0.000	0.000	0.000	0.000
43	0.504	0.000	0.000	0.000	0.000
44	0.799	0.460	0.000	0.000	0.000
45	0.000	0.000	0.000	0.000	0.000
46	1.165	0.000	0.000	0.000	0.000
47	0.000	0.000	0.000	0.000	0.000
48	0.000	0.000	0.000	0.000	0.000
49	0.000	0.000	0.000	0.000	0.000
50	0.000	0.000	0.000	0.000	0.000
51	0.000	0.000	0.000	0.000	0.000
52	0.000	0.000	0.000	0.000	0.000
53	0.000	0.000	0.000	0.000	0.000
54	0.000	0.000	0.574	0.000	0.000
55	0.000	0.000	0.000	0.000	0.000
56	0.771	0.477	0.000	0.000	0.000
57	0.000	0.000	0.000	0.000	0.000
58	0.000	0.000	0.000	0.000	0.000
59	0.000	0.000	0.000	0.000	0.000
60	0.000	0.000	0.000	0.000	0.000
61	0.500	0.488	0.000	0.000	0.000
62	0.504	0.000	0.000	0.000	0.000
63	0.530	0.000	0.000	0.000	0.000
64	0.000	0.000	0.000	0.000	0.000
65	0.000	0.000	0.000	0.000	0.000
66	0.000	0.000	0.000	0.000	0.000
67	0.000	0.000	0.000	0.000	0.000
68	0.000	0.000	0.000	0.000	0.000
69	0.000	0.000	0.000	0.000	0.000
70	0.869	0.000	0.000	0.000	0.000
71	0.465	0.000	0.000	0.000	0.000

Fig. 12 (Cont'd). Insertion Statistics for Word "Zero" for Prototype Phonemes 36 Through 71

that were employed. The overall fuzzy variables are used only to generate fuzzy statistics, and remain constant for all the words. The word fuzzy variables are used only when attempting to recognize the word spoken, and vary for each word in the vocabulary. Some of the reasons for using separate sets of variables to generate statistics and to recognize words include:

1. It is likely that a word's fuzzy variables will be changed often, and if the statistics were not computed from a separate set of variables then they would have to be recomputed each time the word variables are changed.

2. Using separate sets of fuzzy variables allow the effect of different factors to be biased when the statistics are generated. For example, the effect of insertions (defined below) can be minimized when collecting statistics for the purpose of minimizing the number of insertions that occur when a word is scored by the recognition algorithm. Also, the effect of statistics can be eliminated or minimized when generating the statistics.

The reason for using a separate set of word fuzzy variables for each word is that the importance of different factors may vary for different words. For example, one word may have a larger number of pronunciations than another, and therefore the importance of insertions should be different for each word. Another reason for using word dependent variables, is

that it is easier to optimize the variables on a word basis. A short description of each fuzzy variable, that is the same for both word and overall variables, is provided in Appendix A.

There are three types of error statistics that are generated for each word based on its phoneme representations substitution statistics, insertion statistics, and deletion statistics. A substitution "error" occurs when the word phoneme being examined is among the phoneme choices provided by the acoustic processor, or when a phoneme with a sufficiently high possibility of being substituted for the word phoneme (as indicated by the substitution statistics) is among the choices. Note that the term "error" does not necessarily mean that an error actually occurred. If neither of these conditions is met for a given vector, then an insertion error is said to have occurred. A deletion error occurs when a word phoneme is not represented by any vectors in the acoustic output, or when a substitution has not been made for the word phoneme. A word phoneme may be considered deleted regardless of the number of insertions errors that occurred for it. A more formal definition of these errors will be given in the next section.

Figures 10, 11, and 12 provide the format for the deletion, substitution, and insertion statistics, respectively (for the word zero). The deletion statistics simply indicate

the number of times each word phoneme was deleted when attempting to recognize the word's training files. The substitution and insertion statistics indicate the plausibility of any phoneme being substituted or inserted, respectively, for a given word phoneme. As previously mentioned, the manner in which these statistics are computed will be given at the end of this chapter.

Vector Scoring Algorithm

This section presents the vector scoring algorithm used to compute a score indicating the plausibility of a given word phoneme occurring for a vector of the acoustic output. The word phoneme score for each vector i , depends upon the following factors:

Fl_i : the weight assigned to each phoneme by the acoustic processor for each vector i ;

$F2$: the scale factor assigned to each vector by the acoustic processor;

$F3_{w(j)}$: the plausibility of a phoneme being substituted for a given word phoneme, $w(j)$; and

$F4_{w(j)}$: the plausibility of a phoneme being inserted for a given word phoneme, $w(j)$.

The fuzzy sets for each of these factors will now be discussed. Factor Fl_i is represented by the fuzzy set

$$Fl_i = \{P_1/X_{1,i}, P_2/X_{2,i}, \dots, P_c/X_{c,i}, \dots, P_n/X_{n,i}\} \quad (22)$$

where n is the number of phonemes, P_c is the c th prototype phoneme, and $X_{c,i}$ is the grade of membership of P_c determined by the expression

$$X_{c,i} = (\text{weight assigned to phoneme } P_c \text{ by the acoustic processor})/100, \text{ if } P_c \text{ is among the choices provided for vector } i;$$

$$X_{c,i} = 0, \text{ otherwise.} \quad (23)$$

Factor $F2$ is represented by the fuzzy set

$$F2 = \{V_1/SF_1, V_2/SF_2, \dots, V_i/SF_i, \dots, V_t/SF_t\} \quad (24)$$

where V_i is the i th vector of the acoustic output, t is the total number of vectors, and SF_i is the scale factor assigned to vector i by the acoustic processor.

Factor $F3_{w(j)}$ is given by the fuzzy set

$$F3_{w(j)} = \{P_1/S_{1,w(j)}, P_2/S_{2,w(j)}, \dots, P_c/S_{c,w(j)}, \dots, P_n/S_{n,w(j)}\} \quad (25)$$

where $S_{c,w(j)}$ is the grade of membership of P_c in the set given by

$$S_{c,w(j)} = PS_{c,w(j)} / (\text{nospoke} - \text{ndel}_{w(j)}) \quad (26)$$

where $PS_{c,w(j)}$ is the plausibility of phoneme P_c being substituted for word phoneme $w(j)$ (given by the statistics

discussed earlier), nospoke is the number of training files, and $ndel_{w(j)}$ is the number of times word phoneme $w(j)$ was deleted in the training files. The format of $PS_{c,w(j)}$, nospoke, and $ndel_{w(j)}$ was discussed in the previous section.

Factor $F4_{w(j)}$ is given by the fuzzy set

$$F4_{w(j)} = \{P_1/I_{1,w(j)}, P_2/I_{2,w(j)}, \dots, P_c/I_{c,w(j)}, \dots, P_n/I_{n,w(j)}\} \quad (27)$$

where this set is identical to $F3_{w(j)}$ except that $I_{c,w(j)}$ is the grade of membership of P_c in the set given by

$$I_{c,w(j)} = PI_{c,w(j)}/nospoke \quad (28)$$

where $PI_{c,w(j)}$ is the plausibility of phoneme P_c being inserted for word phoneme $w(j)$. $PI_{c,w(j)}$ was also discussed in the previous section.

Using the fuzzy sets for each of the factors discussed above, the vector scoring algorithm can be separated into the calculation of two intermediate fuzzy sets which are then combined to form the fuzzy set used to determine the vector score. The first intermediate set, fuzzy set A_1 , is obtained from sets $F1_1$ and $F2$, and is based solely on the output of the acoustic processor. The resulting fuzzy set is of the form:

$$A_1 = \{P_1/a_{1,1}, P_2/a_{2,1}, \dots, P_c/a_{c,1}, \dots, P_n/a_{n,1}\} \quad (29)$$

where the grades of membership, $a_{c,i}$, are given by the expression

$$a_{c,i} = [1 - ((1 - X_{c,i})/\text{chve})^{\text{chvf}}] \times [1 - (\text{SF}_i/\text{sfe})^{\text{sff}}], \quad \text{for } X_{c,i} > 0$$

$$a_{c,i} = 0, \quad \text{for } X_{c,i} = 0 \quad (30)$$

where $X_{c,i}$ is the grade of membership of P_c in set Fl_i , SF_i is the grade of membership of V_i in set $F2$, and where chve , chvf , sfe , and sff are fuzzy variables used to alter the effect of the factors Fl_i , and $F2$ on the final vector score. Refer to Appendix A for the upper and lower limits of the fuzzy variables.

The second fuzzy set is computed depending on whether a substitution or insertion error occurs for vector i given word phoneme $w(j)$, and depends only on the fuzzy statistics. By definition, a substitution error occurs if the grade of membership, $X_{w(j),i}$ for phoneme $P_{w(j)}$ in set Fl_i is not zero, or if

$$S_{c,w(j)} > \text{sthr} \times S_{w(j),w(j)} \quad \text{for any } c,$$

$$1 \leq c \leq n, \quad \text{in set } F3_{w(j)} \quad (31)$$

where sthr is a fuzzy variable used to determine the "substitution threshold" for the word being examined, and where $S_{w(j),w(j)}$ represents the plausibility of word phoneme $w(j)$ being substituted for itself ($S_{w(j),w(j)}$ is not necessarily

equal to one). If a substitution error does not occur for vector i , then by definition the vector is considered to be an insertion vector (a vector where an insertion error occurs).

If a substitution error occurs, then the second intermediate fuzzy set, $B_{w(j)}$, of the form

$$B_{w(j)} = \{P_1/b_{1,w(j)}, P_2/b_{2,w(j)}, \dots, P_c/b_{c,w(j)}, \dots, P_n/b_{n,w(j)}\} \quad (32)$$

is generated by assigning the grades of membership of the set using the equation

$$\begin{aligned} b_{c,w(j)} &= 1 - [(1 - S_{c,w(j)})/\text{state}]^{**} \text{statf}, \text{ for } X_{c,i} > 0 \\ &= 0, \text{ for } X_{c,i} = 0 \end{aligned} \quad (33)$$

where $X_{c,i}$ is defined previously for set Fl_i , and where state and statf are fuzzy variables used to vary the importance of statistics on the vector score. The fuzzy set which indicates the plausibility of each prototype phoneme replacing the word phoneme $w(j)$, for vector i , is then given by

$$\begin{aligned} SC_{i,w(j)} &= \{P_1/sc_{1,i,w(j)}, P_2/sc_{2,i,w(j)}, \dots, \\ &P_c/sc_{c,i,w(j)}, \dots, P_n/sc_{n,i,w(j)}\} \end{aligned} \quad (34)$$

where

$$sc_{c,i,w(j)} = [(((1-statg) \times a_{c,i}) + (statg \times b_{c,w(j)}))/sube]^{subf} \quad (35)$$

with statg a fuzzy variable used to determine the importance of fuzzy set A_i versus set $B_{w(j)}$ on the vector score, and sube and subf are fuzzy variables used to determine the effect of substitutions on the total word score.

If an insertion error occurred for vector i , then the grades of membership for set $B_{w(j)}$ are computed using the expression

$$b_{c,w(j)} = 1 - [(1 - I_{c,w(j)})/state]^{statf}, \quad \text{for } X_{c,i} > 0$$

$$= 0, \text{ for } X_{c,i} = 0 \quad (36)$$

and the set SC_i is computed using the equation

$$sc_{c,i,w(j)} = [(((1 - statg) \times a_{c,i}) + (statg \times b_{c,w(j)}))/inse]^{insf} \quad (37)$$

where inse and insf are fuzzy variables used to vary the effect of insertions on the total word score.

Given set $SC_{i,w(j)}$, the vector score for vector i ($VS_{i,w(j)}$) given word phoneme $w(j)$, becomes

$$VS_{i,w(j)} = \text{Ave} \{sc_{1,i,w(j)}, sc_{2,i,w(j)}, \dots, sc_{c,i,w(j)}, \dots, sc_{n,i,w(j)}\} \quad (38)$$

where "Ave" indicates that the average of the members of the set is being computed.

Transition Algorithm

Figure 13 illustrates the transition algorithm used to arrive at a word score from the acoustic output. The circles in this figure represent the phonemes in a word's phoneme representation, and the arrows represent transitions from one vector of the acoustic output to the next. A transition can occur regardless of whether a substitution or insertion error occurred for a given vector. The arrows between the circles indicate the possible transitions that can occur between word phonemes. The vector scoring algorithm discussed in the preceeding section is used to determine where the transitions between word phonemes occur.

For each vector, the vector scores for the current and next two word phonemes are computed. If the vector score for the current word phoneme is the highest, then this phoneme remains the current one; however, if either of the next two word phonemes have a higher vector score, then a test must be performed to determine if the vector is a transition between word phonemes or an error. This test consists of averaging the vector scores of each of the three word phonemes for a specified number of following vectors (nsum), and applying one of the following two tests depending on which word phoneme had the highest score.

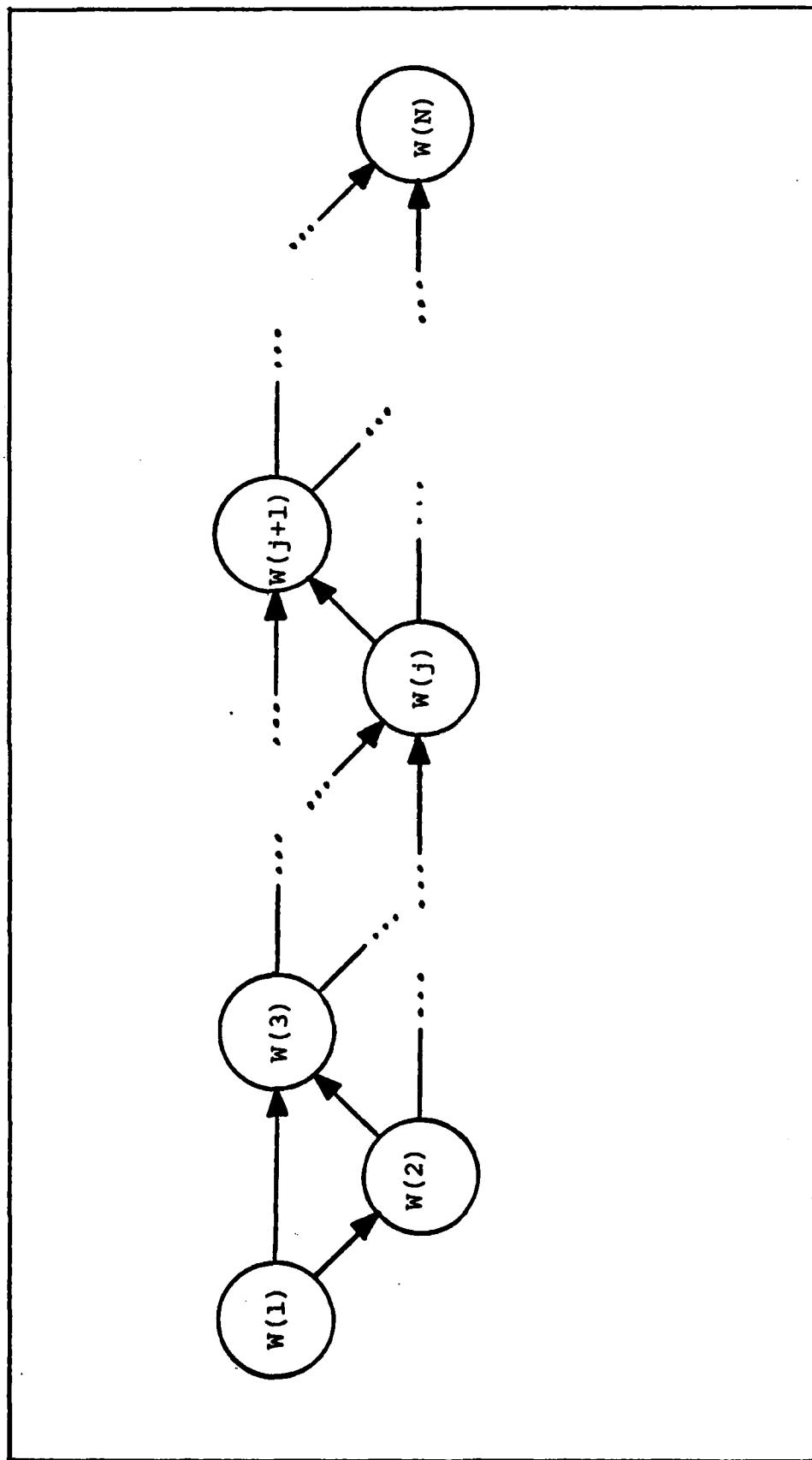


Fig. 13. Transition Diagram Indicating the Possible Transitions That Can Occur Between Word Phonemes

If the word phoneme directly following the current word phoneme had the highest vector score then this test is applied.

Test 1:

$$\text{If } \text{thrle} \times \left(\left(\sum_{k=i}^{i+nsum} V_{k,w(j)} \right) / (nsum+1) \right)^{\text{thrlf}} \geq \left(\sum_{k=i}^{i+nsum} V_{k,w(j+1)} \right) / (nsum+1) \quad (39)$$

then $w(j)$ remains the current word phoneme;

else phoneme $w(j+1)$ becomes the current word phoneme, where i is the number of the current vector, $w(j)$ is the number of the current word phoneme, $nsum$ is the number of vectors to be averaged, $V_{k,w(n)}$ is the vector score computed for word phoneme $w(n)$ for vector k using the vector scoring algorithm, and thrle and thrlf are fuzzy variables used to minimize the possibility of making a transition between word phonemes when an error actually occurred.

If the second word phoneme following the current one had the highest vector score, then the following test is used.

Test 2:

$$\text{If } \text{thrle} \times \left(\left(\sum_{k=i}^{i+nsum} V_{k,w(j)} \right) / (nsum+1) \right)^{\text{thrlf}} \geq \left(\sum_{k=i}^{i+nsum} V_{k,w(j+1)} \right) / (nsum+1)$$

$$\text{and thr2ex} \left(\left(\sum_{k=i}^{i+nsum} v_{k,w(j)} \right) / (nsum+1) \right)^{\text{thr2f}} \geq \left(\sum_{k=i}^{i+nsum} v_{k,w(j+2)} \right) / (nsum+1) \quad (40)$$

then $w(j)$ remains the current word phoneme;

else if

$$\left(\sum_{k=i}^{i+nsum} v_{k,w(j+1)} \right) / (nsum+1) > \text{thrlex} \left(\left(\sum_{k=i}^{i+nsum} v_{k,w(j)} \right) / (nsum+1) \right)^{\text{thr1f}}$$

$$\text{and thr2ex} \left(\left(\sum_{k=i}^{i+nsum} v_{k,w(j+1)} \right) / (nsum+1) \right)^{\text{thr2f}} \geq \left(\sum_{k=i}^{i+nsum} v_{k,w(j+2)} \right) / (nsum+1) \quad (41)$$

then $w(j+1)$ becomes the current word phoneme;

else $w(j+2)$ becomes the current word phoneme,

where thr2e and thr2f are fuzzy variables used to minimize the possibility of making a transition from the current word phoneme to the second word phoneme following it (deleting the phoneme inbetween), when an error actually occurred. Whenever a transition between word phonemes is made, a check is performed to determine if a phoneme was deleted. If a deletion occurred, then a count indicating the number of word phonemes that have been deleted is incremented, and a score based on the deletion statistics is updated using the equation

$$\text{delw}(\text{new}) = \text{delw}(\text{old}) + [(\text{ndel}_{w(j)}/\text{nospoke})/\text{dele}]^{\text{delf}} \quad (42)$$

where delw indicates the word's deletion score based on statistics, $\text{ndel}_{w(j)}$ is the statistic indicating the number of times word phoneme $w(j)$ was deleted in the "nospoke" training files used to collect statistics; and dele and delf are fuzzy variables used to adjust the importance of the deletion statistics on the deletion score.

The process essentially remains the same when there are only two word phonemes remaining, except that the vector score for the second phoneme following the current one is simply considered to be zero (since it does not exist). When there are less than four vectors remaining to be examined in the speech file, the process is slightly modified to discourage deletion errors. This change simply consists of making the word phoneme with the highest vector score the current phoneme rather than performing any further tests.

Total Word Score

The word score obtained for each word is based on the deletion count and deletion score delw given in the last section, and on the vector score, VS_i , obtained for each vector. The total deletion score for a word is given by the equation

$$\begin{aligned} \text{total deletion score} = & 1 - [1 - ((1 - (\text{deletion count}/ \\ & \text{wordlength}) \times \text{dcng}) \\ & + ((1 - \text{dcng}) \times (\text{delw}/\text{deletion} \\ & \text{count}))]/\text{dcne}]^{\text{dcnf}} \quad (43) \end{aligned}$$

where wordlength is the number of phonemes in the word's representation, dcng is a fuzzy variable used to vary the importance of the number of deletions in a word versus the deletion statistics; and dcne and dcnf are fuzzy variables used to alter the effect of the total deletion score on the word score. The quantity

$$(1 - (\text{deletion count}/\text{wordlength}))$$

used in the above equation is used to overcome the problem that arises from varying word lengths for different words. For example, if a word with 10 word phonemes in its representation has 2 deletions, then obviously this word's deletion score should be better than a word with 2 deletions out of only 3 word phonemes.

The word score, WS, for the word being examined then becomes

$$\begin{aligned} \text{WS} = & (\text{delg} \times (\text{total deletion score})) \\ & + (1 - \text{delg}) \times \left(\sum_{i=1}^t \text{VS}_i \right) / t \end{aligned} \quad (44)$$

where t is the total number of vectors, VS_i is the vector score for vector i , and delg is a fuzzy variable which alters the importance of the deletion score versus the average vector score.

After the word scoring algorithm presented in this chapter is applied to each word in the vocabulary, the word chosen as the one spoken is given by the highest word score.

Computation of Fuzzy Statistics

This section presents the fuzzy algorithm used to generate the fuzzy statistics discussed at the beginning of this chapter. Initially, the substitution, insertion, and deletion statistics are set at zero. The word recognition algorithm is then applied to each training file, and the results of this operation are used to generate the statistics. When the recognition algorithm is applied to a file, the following information is obtained for each vector i :

- the word phoneme chosen,
- the error that occurred (substitution or insertion), and
- the vector score.

For example, the results of applying the recognition algorithm to the speech file shown in Figure 2 using the data shown in Figures 8 through 12, is shown in Figure 14. This information is then coordinated with the original training file to create the statistics.

First the weights assigned to each phoneme choice for each vector are combined with the scale factor to derive the fuzzy set A_i , which was defined in the vector scoring section. Then by defining the set

$$SUB_{w(j)} = \{sv_1, sv_2, \dots, sv_k\} \quad (45)$$

as the set of vectors where a substitution error occurred for word phoneme $w(j)$, and the set

VECTOR	=	1	2	3	4	5	6	7	8	9	10
PHONEME	=	21	21	21	21	21	21	21	21	21	70
SCORE	=	9.21E-01	9.17E-01	9.13E-01	9.15E-01	9.20E-01	9.11E-01	8.97E-01	8.96E-01	8.86E-01	8.70E-01
ERROR	=	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST
VECTOR	=	11	12	13	14	15	16	17	18	19	20
PHONEME	=	70	70	70	70	70	70	70	70	70	70
SCORE	=	8.28E-01	7.98E-01	8.09E-01	7.69E-01	7.84E-01	8.08E-01	7.99E-01	8.09E-01	8.53E-01	8.59E-01
ERROR	=	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST
VECTOR	=	21	22	23	24	25	26	27	28	29	30
PHONEME	=	70	70	70	70	70	70	70	70	70	70
SCORE	=	8.63E-01	8.68E-01	8.64E-01	8.62E-01	8.47E-01	8.28E-01	8.75E-01	8.82E-01	9.07E-01	8.88E-01
ERROR	=	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST
VECTOR	=	31	32	33	34	35	36	37	38	39	40
PHONEME	=	70	70	70	70	70	70	70	70	70	70
SCORE	=	8.24E-01	8.10E-01	8.21E-01	8.67E-01	8.87E-01	8.87E-01	8.93E-01	8.98E-01	8.95E-01	8.59E-01
ERROR	=	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST
VECTOR	=	41	42	43	44	45	46	47	48	49	50
PHONEME	=	70	70	70	70	70	70	70	70	70	70
SCORE	=	8.59E-01	8.60E-01	9.06E-01	9.01E-01	8.99E-01	8.76E-01	8.79E-01	7.36E-01	0.00E+00	0.00E+00
ERROR	=	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	SUBST	XXXXX	XXXXX

Fig. 14. Vector Results Obtained by Applying Recognition Algorithm to the Speech File Shown in Figure 2 Using the Data Shown in Figures 8 Through 12

$$INS_{w(j)} = \{iv_1, iv_2, \dots, iv_k\} \quad (46)$$

as the set of vectors where an insertion error occurred for word phoneme $w(j)$, the fuzzy statistics can be computed. Note that the set of vectors where phoneme $w(j)$ was chosen is given by the union of sets $SUB_{w(j)}$ and $INS_{w(j)}$.

If $SUB_{w(j)}$ is the null set, then phoneme $w(j)$ is considered deleted, and the statistic representing the number of times a word phoneme is deleted is incremented as follows

$$ndel_{w(j)} = ndel_{w(j)} + 1 \quad (47)$$

otherwise $ndel_{w(j)}$ remains unchanged.

Defining the fuzzy sets

$$PS_{w(j)} = \{P_1/S_{1,w(j)}, P_2/S_{2,w(j)}, \dots, P_c/S_{c,w(j)}, \dots, P_n/S_{n,w(j)}\} \quad (48)$$

$$PI_{w(j)} = \{P_1/I_{1,w(j)}, P_2/I_{2,w(j)}, \dots, P_c/I_{c,w(j)}, \dots, P_n/I_{n,w(j)}\} \quad (49)$$

as the sets representing the substitution and insertion statistics, respectively, for phoneme $w(j)$ using the training file being examined; the fuzzy statistics for a word can be updated for each training file. These two sets are computed respectively, using the equations

$$FS_{w(j)} = A_{sv_1} \cup A_{sv_2} \cup A_{sv_3} \cup \dots \cup A_{sv_k} \quad (50)$$

$$FI_{w(j)} = A_{iv_1} \cup A_{iv_2} \cup A_{iv_3} \cup \dots \cup A_{iv_k} \quad (51)$$

where \cup indicates the union operation discussed in Chapter 3. The new substitution or insertion statistics are then updated using the equations

$$PS_{c,w(j)}(new) = PS_{c,w(j)}(old) + S_{c,w(j)} \quad (52)$$

$$PI_{c,w(j)}(new) = PI_{c,w(j)}(old) + I_{c,w(j)} \quad (53)$$

where $PS_{c,w(j)}$ and $PI_{c,w(j)}$ were defined in the vector scoring section. Once these statistics are computed for each word phoneme in the word representation, the quantity *nospoke*, indicating the number of training files used to generate the statistics, is incremented.

Any number of training files can be used to generate the statistics for a word. The statistics could also be updated whenever a recognition attempt is made, provided that the actual word is known.

The advantage of using statistics based on the output of the recognition algorithm, rather than on its input, is that errors or peculiarities of the algorithm can be somewhat overcome using this approach.

V. Determination of Word Representations

As noted in Chapter 2, the output of the acoustic processor is extremely variable. Therefore the initial problem was not one of determining the word spoken using a set of known phoneme representations, but was one of first determining the most consistent phoneme representation for each word in the vocabulary. This was necessary since the expected phoneme strings for an utterance did not correspond with the actual strings that were output. Although the fuzzy statistics and variables provide a basis for overcoming this problem, it was found that the accuracy of the recognition algorithm increased substantially when a more consistent phoneme representation was used.

At first it was thought that a more consistent representation for a word could easily be found by examining a number of sample output files for the word; however, it soon became apparent that this was not a simple task. One reason for this is that the output is so variable that no obvious representation exists. Another problem is that the word recognition algorithm attempts to overcome the errors present in a word's representation, making it virtually impossible to determine if a given representation is the best one possible. Because of these problems, an algorithm which automatically determines the phoneme representation for a word was developed.

It was found that, in general, this algorithm generated better word representations than could be generated manually. However, this algorithm does not necessarily derive optimal representations, and further research in this area is recommended.

The determine word representation algorithm first creates an initial representation for the word using the word's training files, and then attempts to improve it using fuzzy statistics generated for this representation. Before an initial representation is created, each vector of each training file is converted into the fuzzy sets A_i , which were defined in the vector scoring section of the last chapter. However, since a number of files are operated on in parallel, another index, g , was added to indicate the training file being examined. The fuzzy set, $A_{i,g}$ is then of the form

$$A_{i,g} = \{P_1/a_{1,i,g}, P_2/a_{2,i,g}, \dots, P_c/a_{c,i,g}, \dots, P_n/a_{n,i,g}\} \quad (54)$$

with $a_{c,i,g}$ computed in the same manner as $a_{c,i}$ was in the last chapter. The initial representation can then be developed using the summation of these fuzzy sets for a specified number of vectors.

The first phoneme of the initial word representation, $w(1)$, is given by the expression

$$w(1) = \text{Max} \left\{ \sum_{g=1}^G \sum_{i=1}^M A_{i,g} \right\} \quad (55)$$

where G is the number of training files for the word being examined, M is the number of vectors to be summed (typically chosen as the number of vectors corresponding to the duration of one phoneme), and $\sum_{g=1}^G \sum_{i=1}^M A_{i,g}$ is a fuzzy set of the form

$$A_{i,g} = \{P_1/d_1, P_2/d_2, \dots, P_c/d_c, \dots, P_n/d_n\} \quad (56)$$

with grades of membership, d_c , given by the equation

$$d_c = \left[\sum_{g=1}^G \sum_{i=1}^M a_{c,i,g} \right] / (M * G) \quad (57)$$

Since the lengths, in vectors, of a word phoneme varies for each training file, it is necessary to find the vector where a transition occurs between word phonemes for each file. To accomplish this, a threshold is used to indicate where a word phoneme ends in a training file. The last vector of the word phoneme is given by the vector, i , satisfying the expression

$$\left(\sum_{k=i+1}^{i+M} a_{w(j),k,g} \right) / M < \text{threshold} \quad (58)$$

with M being a constant indicating the number of vectors to be averaged. Once this vector is determined for the file g , a score is obtained for each prototype phoneme based on the summation of the fuzzy sets, $A_{i,g}$, for the next M vectors, as shown by the equation

$$G_g = \sum_{k=i+1}^{i+M} A_{k,g} \quad (59)$$

Fuzzy set G_g provides a measure of the most likely word phoneme, $w(j+1)$, that follows the current word phoneme, $w(j)$, in training file g . Repeating this operation for all the training files, and summing the resulting fuzzy sets G_g , the next word phoneme of the initial representation, $w(j+1)$, is given by

$$w(j+1) = \text{Max} \left\{ \sum_{g=1}^G G_g \right\} \quad (60)$$

This entire process is repeated until every vector of all the word's training files have been accounted for. When the initial representation is completed, an operation is performed to insure that no two of three consecutive word phonemes are duplicated. This is required since the transition algorithm determines where transitions occur based on the vector scores of the current and next two word phonemes in the representation. If any two of three consecutive phonemes are the same, then the results may become distorted. This problem is alleviated by simply deleting any phoneme in the representation with a duplicate as one of the preceeding two word phonemes.

After an initial representation is obtained, an attempt is made to improve it by using the representation to generate fuzzy statistics. These statistics are generated by applying the word recognition algorithm on the word's training files

as discussed in the previous chapter. At present, there are two separate operations that are used to improve the representation based on the statistics.

The first operation uses the substitution and insertion statistics to improve the representation. After the statistics are generated using the initial representation, the substitution and insertion statistics are combined to form fuzzy sets for each word phoneme of the form

$$O_{w(j)} = \{P_1/o_{1,w(j)}; P_2/o_{2,w(j)}; \dots; P_c/o_{c,w(j)}; \dots; P_n/o_{n,w(j)}\} \quad (61)$$

where

$$o_{c,w(j)} = (PS_c + PI_c)/(2 \times \text{nospoke}) \quad (62)$$

PS_c , and PI_c have been defined in the last chapter. These sets are simply the summation of the corresponding substitution and insertion plausibilities for each word phoneme.

Given the set $O_{w(j)}$ for word phoneme $w(j)$, and defining the sets $O_{w(0)}$ and $O_{w(N+1)}$ as

$$O_{w(0)} = O_{w(N+1)} = \{P_1/0, P_2/0, \dots, P_c/0, \dots, P_n/0\} \quad (63)$$

where N is the number of word phonemes in the representation, a phoneme is inserted before $w(j)$ in the representation if

$$\begin{aligned}
& (w(j) \neq \text{Max}\{O_{w(j)}\}) \text{ and } (O_{\text{Max}\{O_{w(j)}\}, w(j-1)} \\
& \geq O_{\text{Max}\{O_{w(j)}\}, w(j+1)}) \text{ and} \\
& (O_{\text{Max}\{O_{w(j)}\}, w(j-1)} \geq O_{w(j), w(j-1)})
\end{aligned} \tag{64}$$

otherwise if

$$w(j) \neq \text{Max}\{O_{w(j)}\} \tag{65}$$

the phoneme given by $\text{Max}\{O_{w(j)}\}$ is inserted after word phoneme $w(j)$. If neither of the above conditions is met, then no phonemes are added. This process is repeated for each word phoneme $w(j)$, where $1 \leq j \leq N$. When this operation is completed, the new representation is revised to eliminate any duplicates occurring within any three consecutive word phonemes as before, and new statistics are computed using the revised representation.

The second operation used to improve the representation given above, simply eliminates the word phonemes that satisfy the condition

$$\text{ndel}_{w(j)} > (\text{nospoke}/2) + 1 \tag{66}$$

where $\text{ndel}_{w(j)}$ and nospoke have been defined previously. The purpose of this operation is to delete the phonemes in the representation that appear to be invalid. These two operations can be performed iteratively for any specified number

of iterations.

As indicated at the beginning of this chapter, the determine word representation algorithm does not necessarily produce optimum word representations. This algorithm is only a suggested approach, and other algorithms, or variations of this one, should be examined.

VI. Optimization of the Fuzzy Variables

Because of the nature of the word recognition algorithm, it is virtually impossible to determine the optimum values for the fuzzy variables. As explained throughout this thesis, the purpose of the fuzzy variables is to alter the importance of each of the factors used to determine the word that was spoken. One method that is typically used to account for the variations in importance of the different factors, is to assign them fixed values. However this is usually done unknowingly because the factors become embedded in the algorithm's design. For example, scaling the distances between 0 and 100, and providing a scale factor for each vector of the acoustic output (discussed in Chapter 2), to vary the importance of the distances obtained for one vector versus the other vectors, usually would not even be considered. If this were not done, the performance of the recognition algorithm would be much lower. Although there is no way of deriving an optimum set of fuzzy variables to maximize the algorithm's performance, a reasonable set of variables can be chosen by examining the performance of the algorithm for different values of the variables.

The algorithm that was developed to find a reasonable set of word fuzzy variables uses the performance information shown in Figure 9 of Chapter 4. Basically, each word fuzzy

variable is altered within specified limits, and performance data is derived by applying the recognition algorithm to a set of sample speech files. The variable's value and limits are then altered based on this data. It should be understood that this method is only one possible method that could have been employed, and is presented only as an example of the type of algorithm that may be useful in finding a reasonable set of variables for a word.

If variable names are assigned to the performance data of Figure 9 as follows:

nospoke: the number of training speech files that were used to determine the fuzzy statistics for the word;

noattempt: the number of recognition attempts that were made for the word;

nocorrect: the number of times the word was correctly recognized;

avescore: the average recognition score;

avediff: the average difference between the word's recognition score and the next highest score (only computed when word is correctly recognized); and

mindiff: the minimum difference between the word's recognition score and the next highest score (also only computed when word is correctly recognized);

then a figure of merit can be computed which will indicate the performance of the recognition algorithm for a given set of fuzzy variables. The equation used to compute the figure of merit, FM, is

$$\begin{aligned} FM = & FV1 \times (\text{nocorrect}/\text{noattempt}) + FV2 \times (\text{avescore}) \\ & + FV3 \times \text{avediff} + FV4 \times \text{mindiff} \end{aligned} \quad (67)$$

where FV1, FV2, FV3, and FV4 are fuzzy variables subjectively chosen to establish the importance of each of the above factors on the figure of merit ($FV1 + FV2 + FV3 + FV4 = 1$). This figure can then be used to determine whether one value of a fuzzy variable produces better results than another value.

Initially, values, minimum limits, and maximum limits are assigned to the word fuzzy variables that are to be evaluated. Then the figure of merit is computed for this set of variables by applying the word recognition algorithm to a set of sample speech files. One of the variables is then altered using the equation

$$\text{fuzzyvar}(\text{new}) = (\text{fuzzyvar}(\text{initial}) + \text{minlimit})/2 \quad (68)$$

where fuzzyvar is the value of the variable, and minlimit is its minimum limit. The recognition algorithm is again applied to the same set of speech files, and a new figure of merit is computed. If the new figure of merit is greater than the

initial figure of merit, then the variable's maximum limit is set to the variable's initial value, and the variable's current value remains unchanged. However, if the new figure of merit is less than the initial one, then the variable's minimum limit and current value are set to its initial value, and another figure of merit is computed for the variable using the value given by

$$\text{fuzzyvar}(\text{new}) = (\text{fuzzyvar}(\text{initial}) + \text{maxlimit})/2 \quad (69)$$

where maxlimit is the variable's maximum limit. If the figure of merit computed using this value is greater than the initial figure of merit, then the variable's value remains unchanged, and its minimum limit is set to the variable's initial value; otherwise, the variable's maximum limit is set to the variable's current value, and then its value is returned to its initial value. This process is repeated for each word fuzzy variable being evaluated for a given set of words, and continues until the program is aborted.

Since the user has the option of choosing which fuzzy variables to examine, and which words are to be used; he can specify any subset of fuzzy variables for any subset of words to be evaluated in an attempt to maximize the system's performance. It should be noted that the above process is applied to each variable for the complete set of words, and then to the other variables being evaluated, before the

process is again repeated for the same variable.

Since this algorithm attempts to find the local maximum, in terms of performance, for each variable, there is no guarantee that the value generated for the variable is optimum because of the non-linearity of the algorithm. Another problem with attempting to determine the optimum set of fuzzy variables for a word, is that the variables are extremely dependent on the phoneme representation chosen for the word, and on the sample speech files used to compute the figures of merit. Also, as might have been guessed, the algorithm is extremely inefficient, and may require many hours of computer time on a conventional computer system to arrive at a reasonable set of variables.

VII. Results

The software developed to support this research provides numerous options. It has been designed so that the only input required to initialize the system is a set of sample training files for each word in the vocabulary. Since the system is completely independent of: the vocabulary, the acoustic process (provided that phonemes are extracted), and the restrictions placed on the system's input; numerous experiments can be performed to assist in the development of an isolated word recognition system. Some experiments that can be easily performed include:

- determining the effect of increasing, or altering the vocabulary;
- analyzing the system's performance for various groups of speakers;
- examining how the system's accuracy degrades for increasing levels of background noise; and
- evaluating variations of the algorithms employed in the acoustic process, or evaluating the performance of a variety of different acoustic processors which output phoneme strings.

Figure 15 illustrates the general steps required to initialize the recognition algorithm for a given experiment. Once the system has been initialized, results can be obtained by fol-

1. Digitize a number of speech files for each vocabulary word to be used for training (using program AUDIOHIST).
2. Execute programs PHDIST and CHOICES for each speech file to obtain the output shown in Chapter 2 (these programs are given in the appendix).
3. Create the following data files for the experiment being performed:

---SPECH, ---RECOG, ---WFILE, ---SFILE, ---FZOPT

where "---" represents any three characters, and are used to indicate the experiment being performed.

4. Store all the training files in data files ---SPECH, and ---RECOG inputting an arbitrary phoneme representation for each word (this step is accomplished using an option of program LEARN).
5. Execute the "Determine Word Representation" procedure for all words (option of program LEARN), to obtain an initial representation for each word. Experimentation using this representation as a basis may lead to a better representation, and should be performed. Note that this step may be bypassed if valid representations already exist. Whenever word representations are altered or input manually procedure "Initstat" of program LEARN must be executed to initialize the word statistics.
6. Initialize the overall and word fuzzy variables, and execute the "Fuzzy Variable Optimization" procedure, given in program LEARN, to optimize the word variables. These variables can also be optimized manually.
7. Delete and re-create data file ---RECOG to initialize it for recognition purposes.

Fig. 15. Steps Required to Initialize Recognition Algorithm for an Experiment

lowing the steps outlined in Figure 16. These steps provide only one example of the many options that are provided by the software. Refer to the documentation given for program LEARN in the appendices for a discussion of all the available options.

Although similar experiments can be performed with other word recognition algorithms, they are very difficult to accomplish. Other systems are very dependent on the exact acoustic process employed, and usually require detailed knowledge of the features being extracted. These systems also require a high level of accuracy for identifying the features, and their performance degrades rapidly as the number of feature extraction errors increases. The word recognition algorithm developed in this thesis does not depend on the details of the acoustic processor, or features extracted; and degrades gracefully as the number of errors made by the acoustic processor increases. This is accomplished from the fact that the only requirement imposed by this algorithm is that the features output for a given word exhibit some degree of consistency. The success of this algorithm relies solely on its ability to determine what sequences of features are consistent for a word, given a number of sample acoustic outputs for the word. Although the word recognition algorithm was designed to determine this consistency without any human intervention, problems

1. Repeat steps 1 and 2 shown in Figure 15 to obtain a set of speech files to be used for recognition purposes.
2. Store all the files obtained from the previous step in data file --- RECOG (using an option of program LEARN).
3. Execute procedure "Recogall" of program LEARN to obtain recognition results for the files stored in --- RECOG. The results of this operation can be obtained by executing program OUTSTAT. This program provides the information shown in Figures 8 through 12 for each word. Detailed recognition information for each file stored in --- RECOG will also be given in file "RESULTS" after the completion of procedure "Recogall".

Fig. 16. Steps required to obtain recognition results after the steps in Figure 15 have been executed.

associated with the Determine Word Representation and Optimize Fuzzy Variable algorithms make human intervention necessary to obtain the best results.

It was found that the Determine Word Representation algorithm does not always generate a reasonable word representation using an acoustic process that does not produce a consistent sequence of phonemes for a word. Experimentation indicated that better word representations were generated for words with more consistent outputs, as expected. Re-examination of the acoustic outputs for the word "zero", given in Figures 2 through 7 of Chapter 2, will illustrate the difficulty in obtaining a valid word representation. Since it was found that in some instances the Determine Word Representation algorithm produces better representations than those generated manually, the approach that was used to generate a word representation consisted of using the algorithm to derive an initial representation, and manually experimenting with this representation to determine if a better one could be found.

As indicated in Chapter 6, the Optimize Fuzzy Variable algorithm requires many hours of computer time on a conventional computer to generate a reasonable set of variables for a word. Because of this, this algorithm was only employed for a small set of words to generate a preliminary set of variables. These variables were then "optimized" by trial

and error to derive a general set of fuzzy variables that were used for all the words in the vocabulary. Although these variables could have been further "optimized" on a word basis; experimentation indicated that such optimization only provided a minor increase in the system's overall recognition accuracy, and therefore should only be accomplished after a fairly high recognition accuracy is obtained using the same variables for each word. In other words, reasonably accurate results can be obtained using the same variables for each word.

The word recognition algorithm exhibited a unique stability which lessened the difficulty in handling the above problems. It was found that numerous word representations produce similar results. In fact, almost any representation that is chosen by examining sample speech files of a word produces results that substantially exceed chance. It was also found that minor variations of most of the fuzzy variables only had a minor impact on the system's performance. These characteristics are exhibited because of the manner in which the word fuzzy statistics are generated, and because of the statistics impact on the word scores.

To indicate the significance of the word recognition algorithm developed in this thesis, two experiments were performed. For each of these experiments the acoustic processor developed by Karl Seelandt was employed using the 71

prototype phonemes shown in Table I. As indicated in Chapter 2, these prototypes were derived from Seelandt's speech for the words "zero" through "nine". The vocabulary that was chosen consisted of 15 words. These words, and their corresponding phoneme representations, generated using sample speech files spoken by GJM, are shown in Figure 17. Note that many of the sounds occurring in the last 5 words do not correspond to any of the prototypes in the phoneme set, and that the representations chosen for the words "zero" through "nine" do not correspond to the expected representations shown in Table I. The overall fuzzy variables that were used to collect the word fuzzy statistics, and the word fuzzy variables that were used for all the words in the vocabulary were shown in Figure 9.

Experiment 1. This experiment provides an indication of the algorithm's performance when recognizing the speech of a dependent speaker. Statistics were collected for each word using 5 training files spoken by GJM. The recognition algorithm was then applied to 90 speech files (6 for each word) that were also generated by GJM. Of the 90 speech files, 82.2% were identified correctly, with 91.1% of the correct words being among the top 2 choices (the 2 highest scoring words).

Experiment 2. This experiment illustrates the performance of the recognition algorithm for recognizing the speech

<u>WORD</u>	<u>PHONEME REPRESENTATION</u>
ZERO	21 - 70 - 7 - 12 - 62
ONE	33 - 39 - 70
TWO	56 - 70 - 62
THREE	23 - 27 - 70 - 62
FOUR	33 - 34 - 54
FIVE	39 - 7 - 54
SIX	56 - 47 - 1 - 29
SEVEN	12 - 7 - 63 - 62
EIGHT	48 - 57 - 1 - 56 - 62
NINE	63 - 17 - 42 - 62
CCIP	70 - 56 - 57 - 52 - 1
ENTER	21 - 69 - 70 - 63
FREQUENCY	27 - 70 - 1 - 54 - 23 - 1
STEP	12 - 9 - 52 - 1
THREAT	7 - 52

Fig. 17. Vocabulary and Corresponding Word Representations

of several speakers without first training the system for these speakers. The system attempted to recognize the speech of seven male speakers, with different accents, training the system with 10 GJM speech files for each word. The results of this experiment for each speaker are shown in Figure 18. As shown, a total of 53.6% of the speech files were correctly identified, with 76.0% of the correct words being among the top 2 choices.

The above results indicate that the word recognition algorithm developed in this thesis is capable of achieving reasonable levels of accuracy given a very inaccurate input. At present the author is unaware of any other algorithms capable of achieving similar results given similar input. The results of Experiment 2 indicate that there is some consistency in the output of the acoustic analyzer for speakers with different characteristics. The results obtained can easily be improved in a number of ways. For example, the use of more training files should produce more accurate results, and the use of a training set consisting of the speech of a number of speakers should significantly improve the accuracy obtained in Experiment 2. Further experimentation will most likely produce word representations, and fuzzy variables that may significantly increase the accuracy obtained for the above experiments. It can easily be verified that 100% accuracy can be obtained by this algorithm given

SPEAKER	NO. SPEECH FILES	PERCENTAGE CORRECTLY IDENTIFIED	PERCENTAGE AMONG TOP 2 CHOICES
BDB	30	50.2	70.0
DCD	30	56.7	70.7
DEP	10	50.0	90.0
JHC	15	46.7	60.7
MJK	10	60.0	90.0
RLC	15	60.7	86.7
RWH	15	46.7	61.3
TOTAL	125	53.6	76.0

Fig. 18. Results of Experiment 2

a perfect input from the acoustic processor. Therefore as the accuracy of the acoustic process improves, the accuracy of the word recognition algorithm will also improve.

VIII. Conclusions

The word recognition algorithm developed in this research obtains reasonable performance goals despite the inaccuracy of the acoustic processor, and the intrinsic variability of the input speech. This fact implies that it may not be necessary to develop an extremely accurate feature extraction process in the acoustic analyzer to achieve automatic speech recognition. Although this algorithm obviously does not represent the final solution to the speech recognition problem, it definitely suggests a useful approach for developing the required decision process to solve this problem, and perhaps many other pattern recognition problems. The relative success of the recognition algorithm indicates that fuzzy set theory may provide the formalism necessary to overcome the problems that arise when making complex decisions based on a number of variable and error prone factors.

Fuzzy set theory provides a means to deal with the imprecisions involved in making decisions. It allows decisions to be based on a more intuitive level rather than on formal mathematical reasoning. The advantage of this can easily be seen by examining some of the many heuristics used throughout artificial intelligence and pattern recognition which provide the only practical solution to many problems. The fuzzy statistics used in the algorithm enable the algorithm to

automatically adapt to the input based on its actual performance. Since these statistics are collected on a word basis, the need for phonological rules is eliminated. The fuzzy equations developed in this thesis allow a number of factors to be considered in an efficient manner, and enables the importance of each factor to be altered. This aspect of the algorithm is important because it essentially provides a method for the program to dynamically restructure the decision process.

The transition algorithm discussed in Chapter 4 provides a method to determine the plausibility of a word matching a speech file regardless of the number of errors that are made. The advantage of using an algorithm such as this one is that it does not fail when there are unexpected errors.

The algorithm can be used for speaker dependent or independent speech, and can be implemented using any acoustic processor which outputs the features in the format discussed in Chapter 2, regardless of what features are extracted.

The flexibility, adaptability, and restructurability exhibited by this algorithm appear to be similar to the properties that enable humans to perform speech recognition, and suggest that the concepts employed in this thesis may also be useful in many other areas of research.

IX. Recommendations

Recommendations for possible improvements of the proposed word recognition algorithm follow.

1) The algorithm currently computes a score for each word in the vocabulary when evaluating a speech file. For large vocabularies this approach may require an excessive amount of computer time to determine the word spoken. One possible solution to this problem that could be easily implemented is to separate the vocabulary into classes of words. The algorithm could then limit the vocabulary search by first determining the class of words that is most likely to have been spoken, and then compute the scores only for the words in this class. Another solution for decreasing the algorithm's processing time is to use thresholds which would eliminate words from further consideration as soon as a word's score falls below a predefined threshold.

2) Alternate fuzzy equations should be examined which would provide greater flexibility and adaptability. For example, the threshold fuzzy equations discussed in Chapter 3 could be used. More general fuzzy equations should also be examined that would allow the algorithm to determine and vary the relationships of factors with one another. Equations such as these may eventually lead to completely restructurable decision processes that would be able to adapt to any

problem where the relevant factors are known.

3) The transition algorithm discussed in Chapter 4 declares exact vector locations where transitions occur between word phonemes; however, examination of Seelandt's acoustic output indicates that the transitions between word phonemes may be smooth (fuzzy). The transition algorithm should be modified to make fuzzy transitions rather than discrete transitions, and the performance of the modified algorithm should be evaluated to determine if the accuracy of the algorithm is increased.

4) The determine word representation algorithm presented in Chapter 5 does not always derive reasonable phoneme representations. Other algorithms, or variations of this one, should be developed to overcome this problem. An interesting solution to this problem would be to eliminate the need for deriving word representations by relying solely on the word fuzzy statistics. This might be accomplished by simply using a single arbitrary representation for each word to generate the statistics, and by making minor modifications to the algorithm.

5) The fuzzy statistics for each word are generated using a set of training speech files and a set of overall fuzzy variables. The effect of using a large number of training files for each word, or collecting statistics for each recognition attempt should be examined. Also, the use of

individual word fuzzy variables instead of overall variables should be considered for generating the statistics.

6) Because of the inefficiency of the fuzzy variable optimization algorithm described in Chapter 6, other techniques for optimizing the fuzzy variables should be developed.

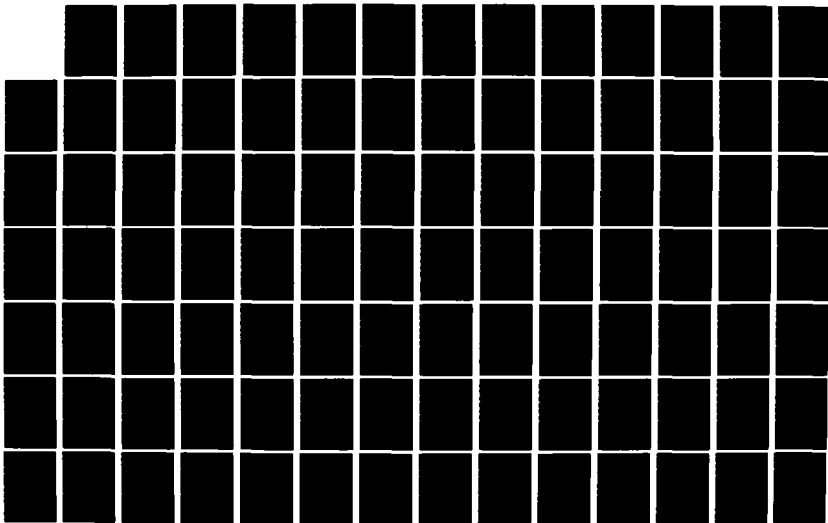
7) It is believed that the accuracy of the recognition algorithm will rise sharply given a more accurate acoustic processor. Certainly, with errorless output from the processor and absolutely consistent speech, it is easy to show that the recognition algorithm would achieve 100% accuracy. What is not known at this time is the rate at which it degrades as the acoustic analyzer makes more mistakes, and as the speech input becomes more variable. Therefore, continuation of research for improving Seelandt's algorithm is highly recommended.

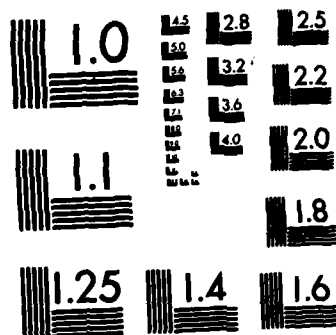
AD-A124 851

ISOLATED WORD RECOGNITION USING FUZZY SET THEORY(U) AIR 2/4
FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING G J MONTGOMERY DEC 82 AFIT/GE/EE/82D-74
F/G 5/8

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

individual word fuzzy variables instead of overall variables should be considered for generating the statistics.

6) Because of the inefficiency of the fuzzy variable optimization algorithm described in Chapter 6, other techniques for optimizing the fuzzy variables should be developed.

7) It is believed that the accuracy of the recognition algorithm will rise sharply given a more accurate acoustic processor. Certainly, with errorless output from the processor and absolutely consistent speech, it is easy to show that the recognition algorithm would achieve 100% accuracy. What is not known at this time is the rate at which it degrades as the acoustic analyzer makes more mistakes, and as the speech input becomes more variable. Therefore, continuation of research for improving Seelandt's algorithm is highly recommended.

Bibliography

1. Baker, Claude A. The Recognition of Words From Phonemes In Continuous Speech. MS Thesis GE/EE/81D-9. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1981.
2. Dubois, Didier, and Henri Prade. "New Results About Properties and Semantics of Fuzzy Set-Theoretic Operators." Fuzzy Sets: Applications to Policy Analysis and Information Systems, edited by Paul P. Wang and S. K. Chang. New York: Plenum Press, 1980.
3. Erman, Lee D. Fredrick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. "The Hearsay II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," Computing Surveys, 12: 213-253 (June 1980).
4. Felkey, Mark A. Automatic Recognition of Phonemes in Continuous Speech. MS Thesis GE/EE/80D-20, Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1980.
5. Fennel, Richard Dean. Multiprocess Software Architecture for AI Problem Solving. PHD Dissertation. Pittsburgh, PA: Computer Science Department, Carnegie Mellon University, May 1975. AD-A015 845.
6. Goodman, S. E., and S. T. Hedetniemi. Introduction to the Design and Analysis of Algorithms. New York: McGraw-Hill Book Company, 1977.
7. Hall, Patrick A. V., and Geoff R. Dowling. "Approximate String Matching," Computing Surveys, 12: 381-402 (December 1980).
8. Kandel, Abraham, and Samuel C. Lee. Fuzzy Switching and Automata: Theory and Applications. New York: Crane Russak and Company, Inc., 1979.
9. Kashyap, R. L., and M. C. Mittal. Spoken Word Recognition Using Statistical and Syntactic Methods in Multi Talker Environment. Interim Report. West Lafayette, IN: School of Electrical Engineering, Purdue University, August 1976.
10. Klatt, Dennis H. "Review of the ARPA Speech Understanding Project," Automatic Speech and Speaker Recognition, edited by N. Rex Dixon and Thomas B. Martin. New York: IEEE Press, 1979.

11. Lesser, Victor R., and Lee D. Erman. "An Experiment in Distributed Interpretation," IEEE Proceedings of the 1st International Conference on Distributed Computing Systems, Huntsville, Alabama: 553-571. (1979).
12. Lesser, Victor R., Richard D. Fennel, Lee D. Erman, and D. Raj Reddy. "Organization of the Hearsay II Speech Understanding System," IEEE Transactions on Acoust., Speech, and Signal Processing, ASSP-23: 11-24 (February 1975).
13. Lowerre, Bruce T. "Dynamic Speaker Adaptation in the Harpy Speech Recognition System," Automatic Speech and Speaker Recognition, edited by N. Rex Dixon and Thomas B. Martin. New York: IEEE Press, 1979.
14. Milne, Robert W. Handling Lexical Ambiguity in a Deterministic Parser. PHD Dissertation. University of Scotland, Edinburgh. Not published.
15. Seelandt, Karl G. Computer Analysis and Recognition of Phoneme Sounds in Connected Speech. MS Thesis GE/EE/81D-53. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1981.
16. Woods, W. A. "Optimal Search Strategies for Speech Understanding Control," Artificial Intelligence, 18: 295-326 (May 1982).
17. Yager, Ronald R. "Multiple Objective Decision-Making Using Fuzzy Sets," Int. J. Man-Machine Studies, 9: 375-382 (1977).
18. Yager, Ronald R. "Satisfaction and Fuzzy Decision Functions." Fuzzy Sets: Applications to Policy Analysis and Information Systems, edited by Paul P. Wang and S. K. Chang. New York: Plenum Press, 1980.
19. Zadeh, L. A. "Fuzzy Sets," Information and Control, 8: 338-353 (1965).
20. Zadeh, L. A. "Toward a Theory of Fuzzy Systems," Aspects of Network and Systems Theory. New York: Holt, Rinehart, and Winston, 1971.
21. Zadeh, L. A. "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes," IEEE Transactions on Systems, Man and Cybernetics, 28-44 (January 1973).

APPENDIX A

Fuzzy Variable Limits

<u>Fuzzy Variable</u>	<u>Lower Limit</u>	<u>Upper Limit</u>
STHR	0.0	infinity
SUBE	1.0	infinity
SUBF	0.0	infinity
INSE	1.0	infinity
INSF	0.0	infinity
DELE	1.0	infinity
DELF	0.0	infinity
DELG	0.0	1.0
DCNE	1.0	infinity
DCNF	0.0	infinity
DCNG	0.0	1.0
SFE	1.0	infinity
SFF	0.0	infinity
CHVE	1.0	infinity
CHVF	0.0	infinity
STATE	1.0	infinity
STATF	0.0	infinity
STATG	0.0	1.0
THR1E	0.0	infinity
THR1F	0.0	infinity
THR2E	0.0	infinity
THR2F	0.0	infinity

APPENDIX B

Recommended Approach for Designing A Speech Understanding System

A speech understanding system (SUS) is a system capable of recognizing speech, and interpreting it to perform some task. A block diagram of a SUS is given in Figure 19. As shown, speech enters the system through a microphone, and is input into an acoustic analyzer. The acoustic analyzer extracts features, such as phonemes, from the speech, and outputs the features to the word sequence hypothesizer. The word sequence hypothesizer determines the most probable word sequence that occurred from the sequence of features. The word sequences are then analyzed by the sentence analyzer which determines the task to be performed. The sentence analyzer also provides information to the word sequence hypothesizer to eliminate the generation of invalid word sequences. To perform these tasks, the sentence analyzer uses the following information.

1. Phonetics. Phonetics is used to account for the different pronunciations of words based on the context of the surrounding words, and on typical variations that are encountered.
2. Syntax. Syntax is used to determine the validity of a word sequence using a set of grammatical rules.
3. Semantics. Semantic information is used to determine

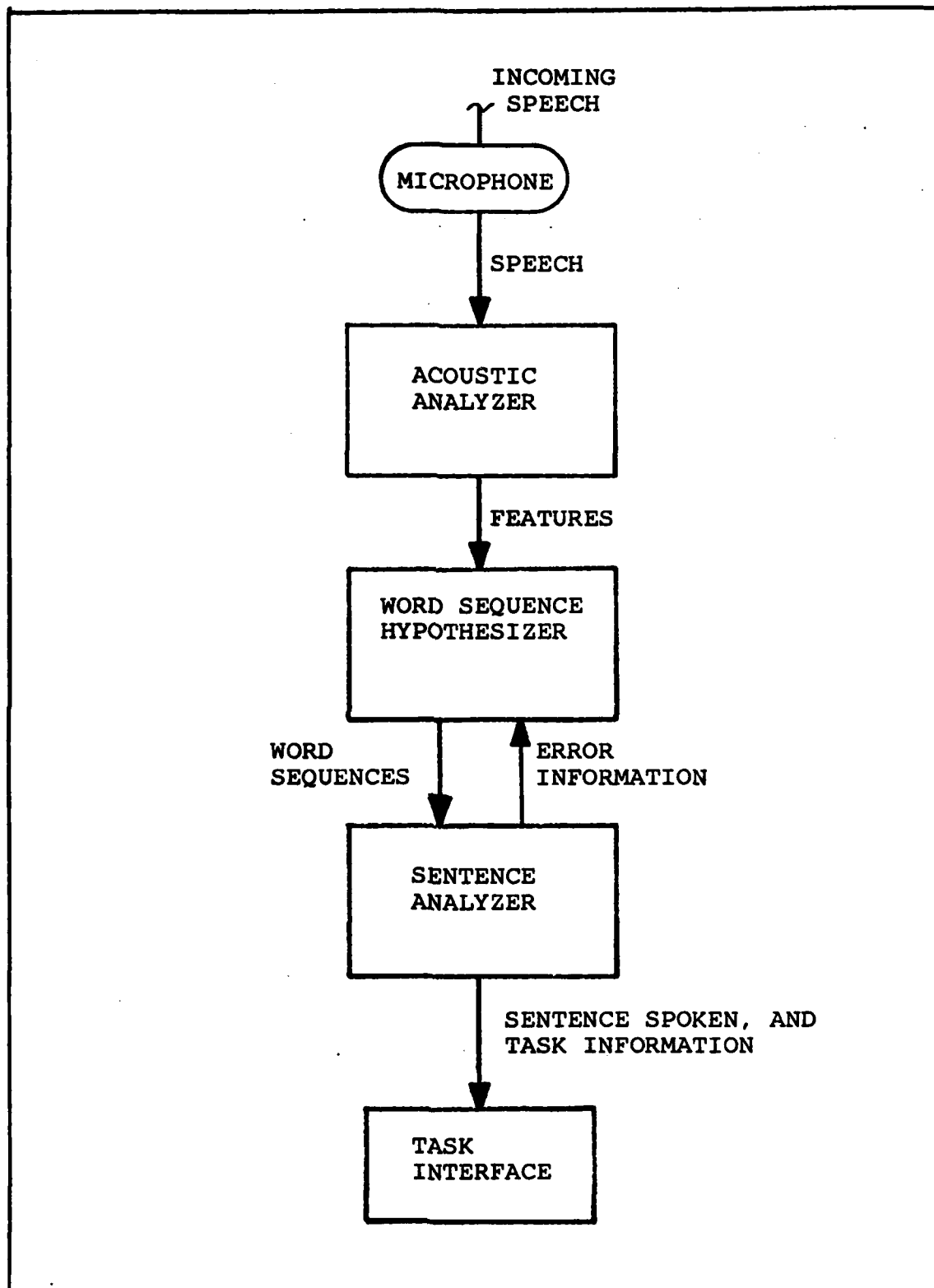


Fig. 19. Block Diagram of a Speech Understanding System

the meaning of a sequence of words.

4. Pragmatics. Pragmatics is the knowledge of the context of the possible tasks, and what has been said previously.
5. Prosody. Prosodic information involves the interpretation of stress and intonation in terms of the syntactic and semantic structure of a word sequence.

The purpose of this appendix is to suggest a general approach for developing a virtually non-restricted speech understanding system. This is accomplished by recommending a possible word sequence hypothesizer, and by suggesting a method for incorporating the above information required by the sentence analyzer. Before an approach can be established, the functional requirements, and possible restrictions that can be employed in realizing a general SUS, must be discussed.

Functional Requirements and Possible Restrictions

Present speech understanding systems employ many different methods to recognize and interpret speech. To achieve practical performance goals, all of these systems impose severe restrictions. Although most of these restrictions must be applied to some degree, they presently limit the usefulness of SUS's to very few applications. A discussion of each of the major restrictions is given below.

1. Mode of Speech

This restriction concerns how close the SUS is to recog-

nizing natural conversational speech. Some systems can only recognize isolated words, i.e. the speaker must pause between each word in a sentence. Other systems are capable of recognizing connected speech, but require the speaker to be trained to speak in an unnatural manner.

2. Speaker Variability

Speaker variability refers to the dialect, contextual, and acoustic variations among speakers. Dialectic variability occurs from the differences in the pronunciation of words among speakers. Contextual variability involves changes in the pronunciation due to the combinations of words. Acoustic variability results from vocal tract differences among speakers. Any or all these types of variability may occur when the speaker is changed. An ideal system's performance would not be effected by speakers that have diverse speaker characteristics. However present systems may be adversely affected by changing speakers. It should be noted that people often have trouble understanding others with different speech characteristics.

A system's speaker variability is based on the variety of speakers allowed, and on whether or not the system, or speaker, must be trained. Some systems only allow speakers which have certain characteristics. For example, the system may require the speaker to be male, or require the speaker to have a particular dialect. Other systems can only recognize

speech from certain speakers, and tune the system to a new speaker by requiring the speaker to repeat a set of training sentences. Perhaps the best method of adapting the system to a speaker is to use a dynamic tuning algorithm, where the system automatically tunes itself to a new speaker (Ref 13).

3. Language

Assuming that English is the required language, the system language can be natural English, a language similar to English, or a command language. An English-like language places restrictions on the structure of the sentences, or number of different sentences allowed, and usually depends on the task being performed. A command language is highly dependent on the task being performed, and consists of a very small set of phrases.

4. Environment

The environment of a SUS is usually dependent on the task to be performed. For example a system to be used in an office would require less tolerance to noise than a system to be used in a jet aircraft. A reasonable environment for an initial SUS would be a computer room, however, some researchers believe that a sound proof room should be used for initial system development.

5. Vocabulary Size

Ideally the vocabulary of a SUS should be unlimited, however the size of the vocabulary effects performance drastically.

A reasonable vocabulary would probably consist of 3,000 to 10,000 words. Other than limiting the size of the vocabulary, the vocabulary can be restricted by selecting words for phonetic discriminability, i.e. selecting words to make the task of matching acoustic segments to the words easier. If the vocabulary is selected with this in mind, the ability of the system to adapt to changes in its task would be lessened.

6. Average Branching Factor

The average branching factor (ABF) is the average number of words that can follow an initial sequence of words as defined by the grammar. Usually a system's performance increases as the ABF decreases, as shown in Figure 20 for the HWIM, SDC, Hearsay II, and Harpy speech understanding systems (Ref 3:243). The reason for this is that lower ABFs substantially limit the number of possible words that can occur after an initial sequence. Thus the number of searches that must be performed to determine the next word in a sequence is reduced. Since this factor can only be reduced by placing additional constraints on the grammar, it is perhaps the best measure of the constraints placed on a system.

7. Task Orientation

This restriction concerns the complexity of the tasks performed by the system, and the dependence of a system on the task. The complexity can range from simple tasks, such

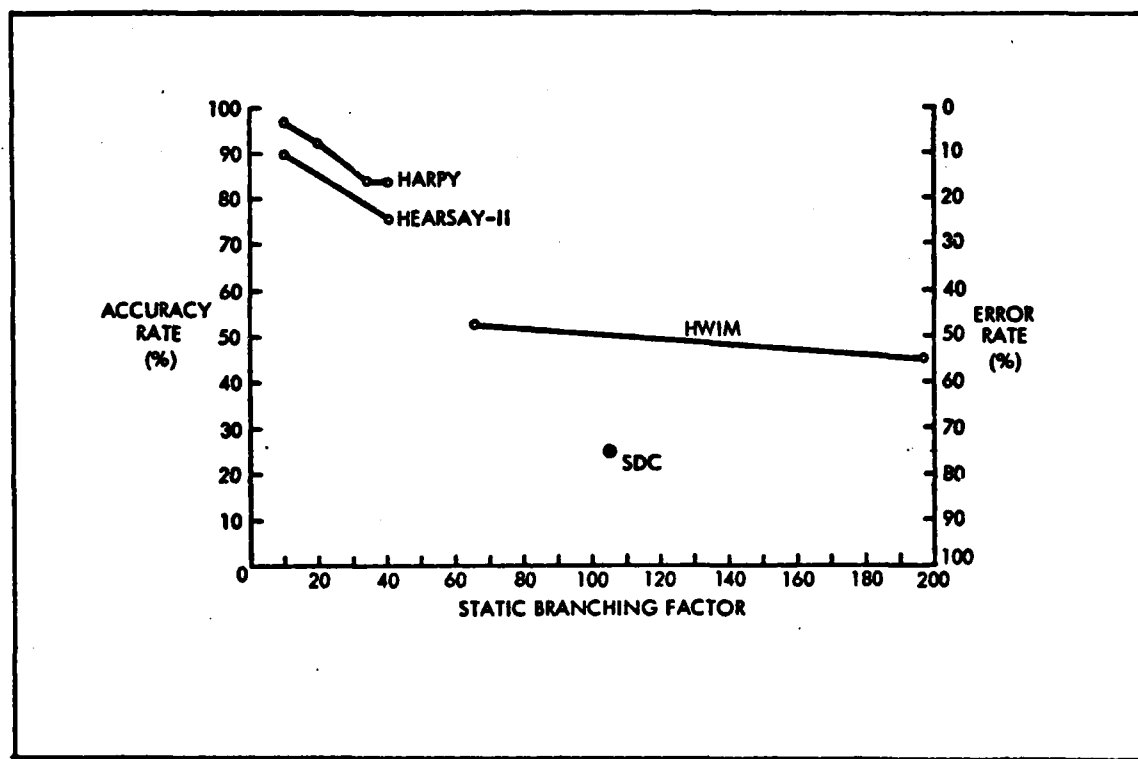


Fig. 20. Effects of Average Branching Factor on Recognition Error Rate (Ref 3:243)

as moving a small number of objects in a simple environment, to very complex tasks which may involve a large number of simpler tasks. The degree of dependence of a SUS on the task can dramatically effect the performance of the system, and is closely related to the other restrictions placed on the system. For example, if a simple task, such as moving blocks, is to be performed, the language necessary would only have to consist of a few simple sentences, and the vocabulary would only have to include a few words. A low ABF is a sign that the system is task dependent.

Taking into consideration the above restrictions, and other factors, the functional requirements for a reasonable SUS are given below.

1. The mode of speech should be natural conversational speech.
2. The system should be capable of accurately recognizing the speech of a number of various speakers, i.e. speakers of both sexes, and speakers with different dialects.
3. The language should be natural English.
4. The system should be capable of handling a large number of different tasks, or should be capable of being easily changed to perform various tasks.
5. The size of the vocabulary should be approximately 3,000 words. A system with a vocabulary of this size will be capable of performing a large range of tasks.
6. The system should operate efficiently in environments with noise levels equivalent to those of the intended applications.
7. The system should have accuracy rates that are very

high. The accuracy should be based on the percentage of sentences interpreted correctly, since many errors may not effect the task to be performed. If a rejection criterion is used, the number of rejections should be small.

8. The system should operate in real time.
9. The system should be flexible in that it should be able to adapt changes in the task, and small variations in the algorithms employed.
10. The system should be reliable in terms of the mean time between failures.
11. The system should be able to determine when one of its components is not functioning properly. It would also be desirable to be able to bypass malfunctioning components.

At present no systems meet these requirements. To further complicate the development of a general-purpose SUS, is the fact that the trade-offs involved among these requirements make it difficult to compare the different architectures of present SUS's. For example, a system with a high task dependence usually obtains more accurate results, and operates much faster than a system with less dependence on the task being performed. Therefore it would appear that the faster and more accurate system is better, which may not be true. Obviously there are many practical applications for SUS's which do not meet the above requirements, and place severe restrictions on the task; however, there are not any existing speech understanding system's other than ones in the experimental stage.

Recommended Approach

The purpose of this section is to present an approach for developing a general speech understanding system which meets the functional requirements given in the preceeding section. This approach is designed to provide many parallel areas of research that can be considered independently. In fact it is strongly recommended that most of the areas be researched simultaneously. The method for developing an SUS should be based on the application of restrictions in a logical manner that will eventually lead to a virtually unrestricted SUS. The 7 research steps that will be presented to achieve this goal are shown below.

1. Development of a Feature Extraction Process.
2. Development of an isolated word recognition algorithm.
3. Development of a connected speech algorithm.
4. Incorporation of phonetic knowledge.
5. Incorporation of syntactic and semantic information.
6. Incorporation of pragmatics and prosody.
7. Development of a SUS computer architecture.

Although the above steps suggest a sequential approach, it is recommended that these steps be independently researched and developed. The algorithms suggested in the following discussion of each step are designed to impose the least number of restrictions necessary to realize a general SUS.

1. Development of a Feature Extraction Process

There are basically two type of feature extraction algorithms: word based, and phoneme based algorithms. Word based algorithms use the actual words as the features, and attempt to identify to the word that was uttered directly from the speech. Because of memory and other constraints, these algorithms severely limit the size of the vocabulary, and tend to limit the flexibility of the entire system. Phoneme based systems, on the other hand, potentially do not place such restrictions on the system, and therefore provide much greater flexibility. Also, more flexibility is obtained by these systems since they can be designed completely independent of the other processes required to understand speech. The word recognition algorithm developed in this thesis verifies this fact because it can be used with any feature extraction process that outputs strings of phoneme guesses. The acoustic process developed by Karl Seelandt appears to represent the type of algorithm that should be used for extracting phonemes.

The only restrictions that must be overcome in Seelandt's algorithm are the "speaker variability" and "environment" restrictions discussed earlier. Initial results obtained from this algorithm indicate that future improvements of this process may eventually overcome these restrictions. Currently, several researchers are attempting to make these improvements, and continued research in this area is recommended.

As shown in this thesis, the accuracy of the acoustic process does not have to be as high as might be expected to determine the words spoken. Because of this fact, and because perfect feature extraction is virtually impossible, the feature extraction research should be conducted in parallel with the remaining areas of research to be discussed in this section.

2. Development of an Isolated Word Recognition Algorithm

Before attempting to recognize continuous speech, it is logical to first develop an isolated word recognition algorithm. Unlike the previous step, an approximate accuracy goal for this algorithm can be established as the accuracy obtainable by humans for isolated word recognition. The achievement of this accuracy will be dependent on the decision scheme used, and on the accuracy of the acoustic analyzer.

The word recognition algorithm developed in this thesis is capable of achieving reasonable accuracy for a small vocabulary despite the inadequacy of the feature extraction algorithm. Both of these algorithms must be improved considerably to obtain the same accuracy obtained by humans. Since the only restriction that should be placed on the word recognition algorithm is the size of the vocabulary, it is recommended that the initial vocabulary be small, and increased slowly as the recognition accuracy increases.

The recognition algorithm presented in Chapter 4 offers a number of advantages over existing algorithms used to determine the word spoken. One advantage not previously discussed is that the number of operations required to determine the word spoken is equal to the number of words in the vocabulary times the number of operations to score one word. Therefore an increase in the vocabulary size does not significantly effect the algorithm's execution time. However, the time required to execute the algorithm in its current state is extensive. Since it will be necessary to continually experiment with variations of this algorithm to improve it, it would be advantageous to minimize the time required to execute the algorithm. This can be accomplished by optimizing the algorithm's code, separating the words into word classes (as discussed in Chapter 9), and by developing a specialized computer system for implementing the algorithm.

Since the same operations are performed to score each word, a computer architecture can easily be developed which would decrease the execution time. Figure 21 illustrates a block diagram of a multiprocessor computer system that would score a number of words in parallel, and that would still allow major modifications to be made in the algorithm. As shown, the acoustic processor outputs the features to each of the N word processors. The supervisor then assigns each word processor a word from the vocabulary to score. After a

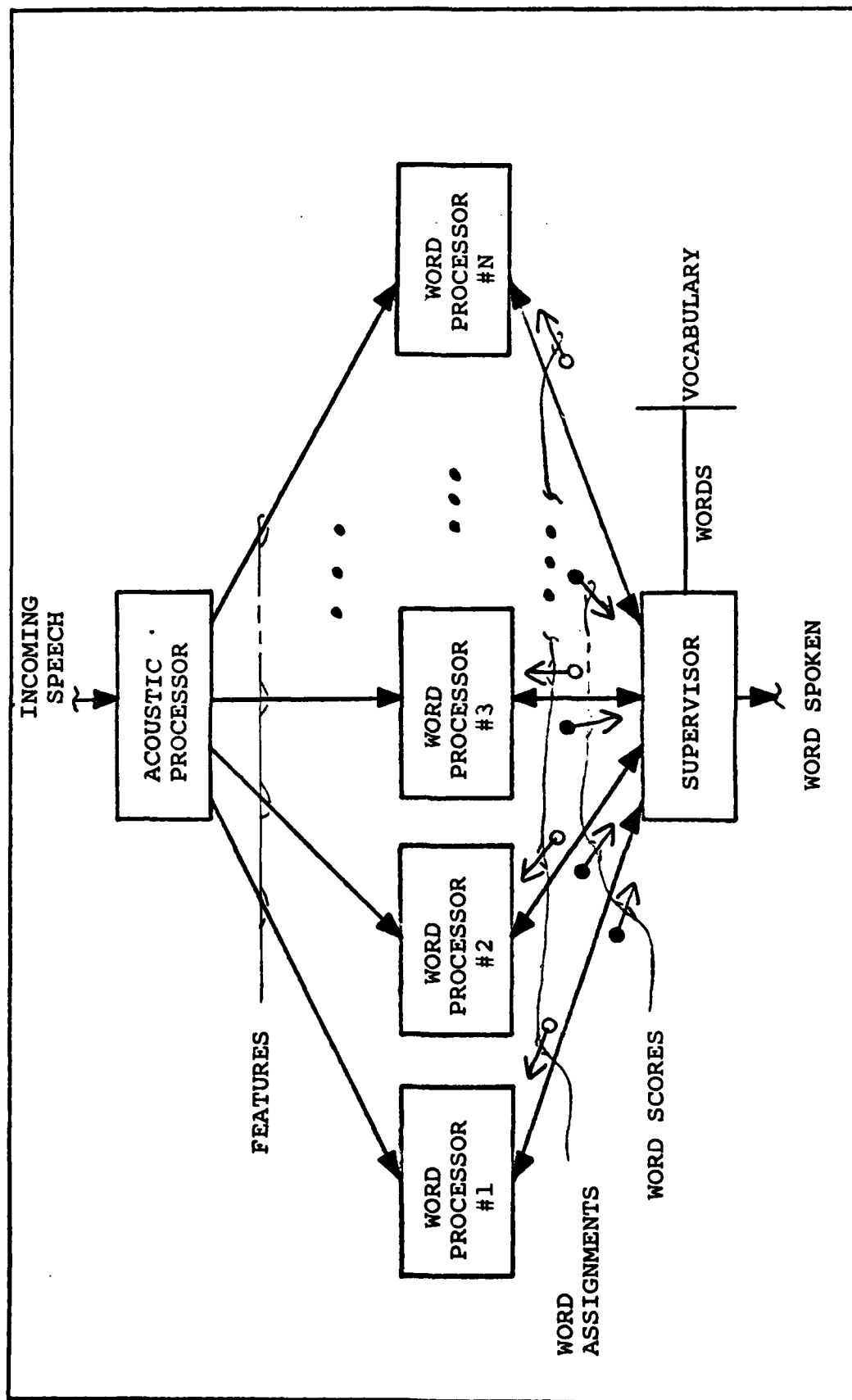


Fig. 21. Block Diagram of a Possible Computer Architecture for Implementing the Word Recognition Algorithm.

word processor scores a word, it returns the word score to the supervisor, and waits for the next word. This process is repeated until all the words have been scored. When this occurs the supervisor then determines which word was spoken based on the word scores received from the word processors.

It must be understood that the purpose of increasing the recognition algorithm's efficiency is to aid the researcher in determining what modifications should be made to the algorithm, and not to achieve real time performance.

3. Development of a Connected Speech Algorithm

Once the isolated word recognition algorithm obtains a reasonable accuracy, this algorithm should be modified to recognize words that appear in connected speech. This should be accomplished by first expanding the isolated word recognition algorithm to locate words in a word string. One possible method for doing this involves the use of thresholds, and the concept of fuzzy word locations.

Thresholds can be used to establish potential beginning and end points for words appearing in a word string. This can be accomplished by requiring word's to have scores above a predefined threshold before they are considered. Fuzzy word locations allow a word's end points to be fuzzy, i.e. the exact location of a word does not have to be specified (Ref 16:302-303). This strategy eliminates the possibility of examining duplicate word hypothesis for the same word in

similar locations, and allows word phrases to be formed without determining the exact word boundaries. Also, errors and overlapping that occur between words can easily be overcome using this approach.

When reasonable accuracies are obtained for determining the possible words, and locations of these words, in a word string; an algorithm which hypothesizes word sequences should be developed. An outline of a possible word sequence hypothesizer, based on fuzzy word locations is shown in Figure 22. As an example of the algorithm, consider the words "one" through "seven", and assume that the possible end points of these words in a word string, shown in Figure 23, were obtained from applying step 1 of the algorithm. Execution of step 2 of the algorithm will then produce the tree shown in Figure 24, and the application of steps 3 through 6, with X equal to three, will produce the possible sequences given in Figure 25.

After all the possible word sequences are produced, each sequence can be assigned a score by summing the fuzzy word scores of each word in the sequence. Many of the hypothesized sequences can then be eliminated based on a predetermined threshold, or by only considering the top choices. Final sentence scores can then be obtained by combining the word representations and word statistics (used in the word recognition algorithm) of all the words in a given sequence to

1. Find all possible word matches in word string using thresholds to determine end points. End points should be chosen to minimize the word lengths.
2. Generate a tree using inorder traversal (Ref 6:243) based on the beginning vector of each word.
3. Form a sentence by traversing the tree inorder, choosing the next word in the sentence as the first word satisfying the expression

$$(\text{Beginning of next word}) \geq (\text{end of last word} - X)$$
 and repeat until no more words satisfy the above expression. The variable "X" in the above expression is used to account for possible overlapping of words.
4. Eliminate last word of previous sentence, and repeat step 3 until the following expression is true:

$$(\text{Beginning of next word}) \geq (\text{end of word being replaced})$$
5. Repeat step 4 by eliminating last word examined until

$$(\text{Beginning of next word}) \geq (\text{end of word being examined})$$
6. Repeat step 5 until the first word has been examined.

Fig. 22. Possible Algorithm for Generating Sentence Hypotheses

WORD	BEGINNING VECTOR	ENDING VECTOR
ONE	1	10
TWO	5	15
THREE	10	20
FOUR	15	25
FIVE	20	30
SIX	25	35
SEVEN	30	40

Fig. 23. Result of Applying Step 1 of Fig. 24

WORD	BEGINNING VECTOR	ENDING VECTOR
ONE	1	10
TWO	5	15
THREE	10	20
FOUR	15	25
FIVE	20	30
SIX	25	35
SEVEN	30	40

Fig. 23. Result of Applying Step 1 of Fig. 24

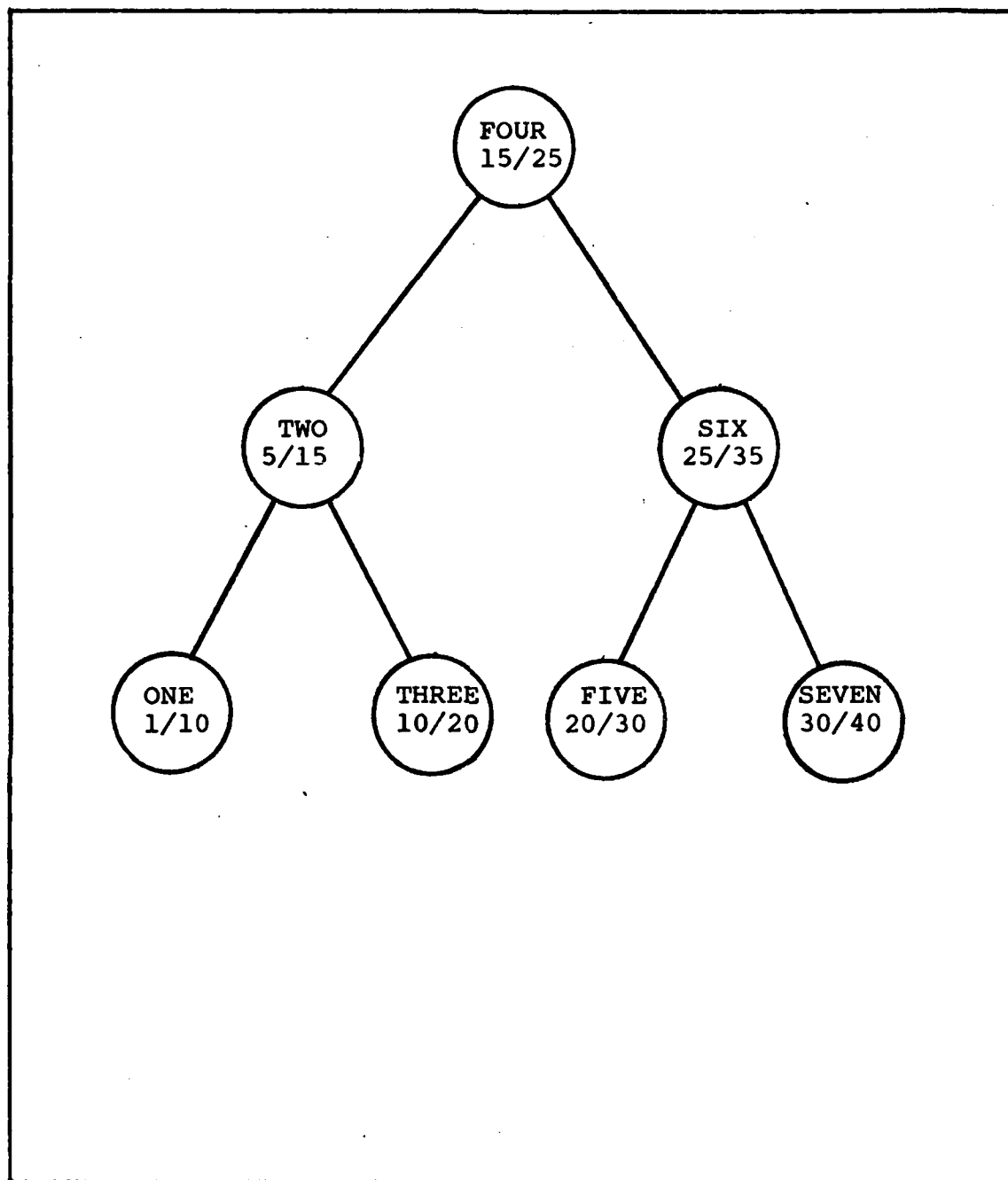


Fig. 24. Tree Resulting from the Application of Step 2 to the Data Shown in Fig. 23. Numbers under each word indicate the word's beginning and end points.

SENTENCE NO.	POSSIBLE SENTENCES
1	ONE-THREE-FIVE-SEVEN
2	ONE-THREE-SIX
3	ONE-FOUR-SIX
4	ONE-FOUR-SEVEN
5	TWO-FOUR-SEVEN
6	TWO-FOUR-SEVEN
7	TWO-FIVE-SEVEN

Fig. 25. Results of Applying Steps 3 Through 6
of Fig. 24 with X Equal to Three

form a phrase, and by applying a modified version of the isolated word recognition algorithm which would treat the phrases as words. The phrase with the highest score would be the one chosen as the sequence actually spoken.

4. Incorporation of Phonetic Knowledge

The phonetic knowledge to be incorporated at this stage, refers only to the phonetic information for the errors that occur between words when they are spoken together. Although this information can be incorporated after the algorithm developed in the previous step achieves reasonable accuracy, it is recommended that this knowledge be incorporated as soon as an initial connected speech algorithm is developed. There are two major reasons for this approach: 1) the accuracy of the connected speech algorithm may be significantly affected by the errors that occur between words; and 2) extensive modifications of the connected speech algorithm may be required to incorporate this knowledge.

Phonetic information can be incorporated by implementing a set of phonological rules, or by using statistics similar to the word statistics used in this thesis. For the reasons presented in Chapter 4, statistics for the errors that occur between words would be preferable; however, because of memory constraints, and difficulties which were not encountered when collecting isolated word statistics, this approach may not be practical.

5. Incorporation of Syntactic and Semantic Information

Only the following restrictions should be considered in the previous steps:

1. mode of speech;
2. speaker variability;
3. environment; and
4. vocabulary size.

This step must consider the remaining restrictions (language, ABF, and task orientation). The simplest and most restricted method for incorporating syntactic, and semantic knowledge is to simply limit the number of possible sentences that can occur. This is exactly what was done in the Harpy SUS developed at Carnegie-Mellon University (Ref 10). The Harpy system employs a state transition network which is formed from all the possible pronunciations of a set of allowable sentences. This approach was taken to increase the SUS performance, and to overcome the lack of practical decision algorithms capable of producing reasonable sentence hypotheses. Obviously the Harpy is extremely inflexible, and can never be used as the basis of a general SUS.

Many researchers claim that humans must also rely on severe restrictions at this level to understand speech; however the author firmly believes that although these restrictions are indeed used extensively by humans, the importance of them have been over-estimated. Syntactic and semantic information should be incorporated without placing unnecessary

restrictions on the system. Restrictions should only be used to aid in the development of the general algorithms. For example, a limited vocabulary could be used to limit the time required to experiment with the algorithms being developed, but the algorithms should remain independent of the exact vocabulary chosen.

Before semantics is considered, a procedure which parses an error-free sentence using only syntactic information should be developed. Such an algorithm has been developed by Milne (Ref 14). This algorithm should then be modified to determine if a given sentence is valid, and then incorporated with the word sequence hypothesizer to eliminate invalid word sequences as they occur, rather than considering them further.

As the syntactic algorithms are developed, semantic algorithms which determine the meaning of a sentence using syntax and the meanings of words, should be developed. It may be advisable to develop this algorithm by initially placing restrictions on the task to be performed, and then by expanding the algorithm to handle more complicated tasks; however the algorithm should remain independent of the specific semantics involved. Fuzzy set theory may prove to be invaluable in determining which task (or tasks) is to be performed when the action required is unclear, or when the task requires the system to make complex decisions. After

this algorithm is developed, it should be modified to eliminate invalid sentences that were not eliminated by the syntactic algorithm, and also incorporated in the word sequence hypothesizer.

6. Incorporation of Pragmatics and Prosodics

Although pragmatic and prosodic knowledge are completely different, both of these knowledge sources can be used to eliminate ambiguities that arise in the syntactic and semantic algorithms developed in the previous step. This information can also be used to increase the system's performance. Since this information is not vital for obtaining a reasonable SUS, it should not be employed until after reasonable success has been attained by the SUS. Implementation of pragmatic or prosodic knowledge at an earlier stage of development will complicate the generation of the more vital algorithms, and will consume needlessly many hours of the researchers time.

7. Development of a SUS Computer Architecture

The practical implementation of the necessary algorithms for speech understanding will most likely require a specialized computer architecture to obtain real time performance. Although standard computer systems should be used where possible in the development of these algorithms, it is illogical to constrain a SUS to operate on a general-purpose computer system.

To design or choose an appropriate computer system for

a speech understanding system, the following factors must be considered.

1. Performance. The architecture chosen must be capable of achieving real time operation. The algorithms developed for speech recognition must be designed with the expectation that only reasonable advancements in computer technology will be made in the near future, and not with the hope that major technological breakthroughs will be made.

2. Flexibility. The architecture should place a minimum number of constraints on the speech understanding algorithms. The system should remain as flexible as possible to allow the algorithms to be modified easily. This aspect of the computer system is exceptionally important since a flexible architecture will enable researchers to use the system to aid in the design of the required algorithms.

3. Reliability. The computer system must be reliable. Excessively complex systems tend to be less reliable than simple ones, and therefore should be avoided.

4. Fault Tolerance. This aspect concerns the interaction of the computer system's components to identify components that are malfunctioning, and considers the system's ability to distribute the functions of the malfunctioning component while minimizing the effects on performance. Fault tolerance also refers to how well a system's performance and accuracy degrades as different components malfunction.

5. Cost. Obviously the cost of the final system is very

important. Although every attempt should be made to minimize the cost, it should be anticipated that the expense of developing an initial system may become very high. Therefore, before any attempt is made to implement a proposed architecture, detailed simulations of the system should be accomplished.

Before attempting to design or choose a SUS architecture, the Hearsay II SUS developed at Carnegie-Mellon University should be examined (Refs 5, 11, and 12). Considering the above factors, the Hearsay II undoubtedly represents the most noteworthy achievement of a SUS architecture. When examining this system, emphasis should be placed on its architecture, and not on the methods employed to solve the speech understanding problem. Because of the complexity of this system, it will not be discussed in this appendix, and the reader is encouraged to see the references.

Remarks

The approach presented in this appendix is intended to provide future researchers with a general guideline for developing a speech understanding system. It is very important that any research in this area, other than feature extraction research (step 1), be performed with an understanding of the entire problem, and not just segments of the problem. The recommended steps are designed to indicate a number of parallel research efforts that can eventually be combined in a logical manner to realize a general SUS.

APPENDIX C

Computer Program Structure Charts

This appendix contains structure charts for program LEARNWORD shown in Appendix D. The format of the charts that appear in this appendix is shown in Figure 26, and a general presentation of structure charts can be found in Structured Analysis written by Victor Weinberg.

The main purpose of this appendix is to illustrate the physical structure of program LEARNWORD. The structure charts that follow should be used in conjunction with the program's internal documentation to obtain a thorough understanding of the program.

It should be noted that programs LEARNWORD and OUTSTAT, shown in Appendix D, are written in Rational Data Systems, Incorporated Pascal, and implement some non-standard functions. The remaining programs are written in Data General Corporation Fortran V.

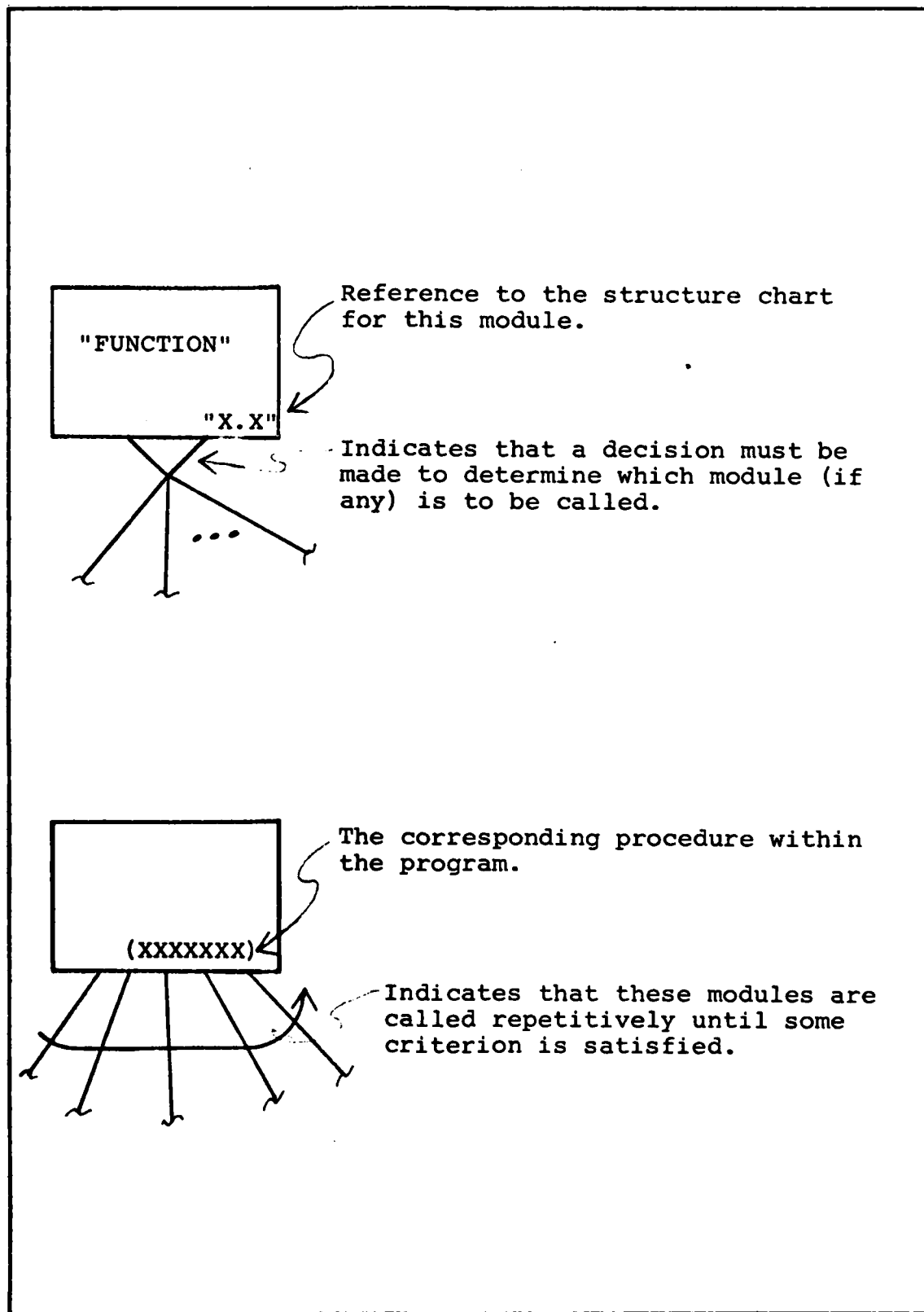


Fig. 26. Legend for the Structure Charts

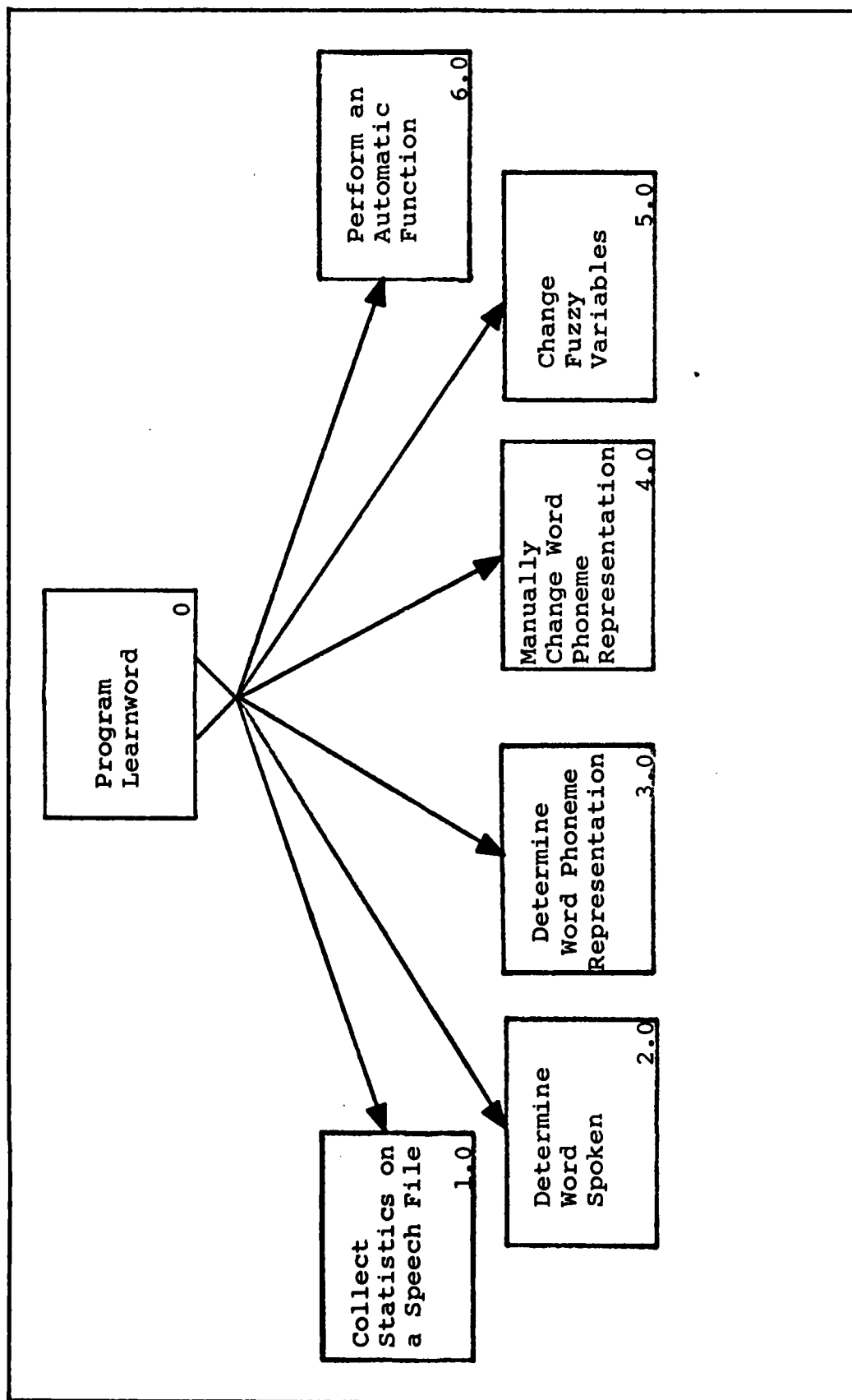


Fig. 27. Structure Chart 0: Program Learnword

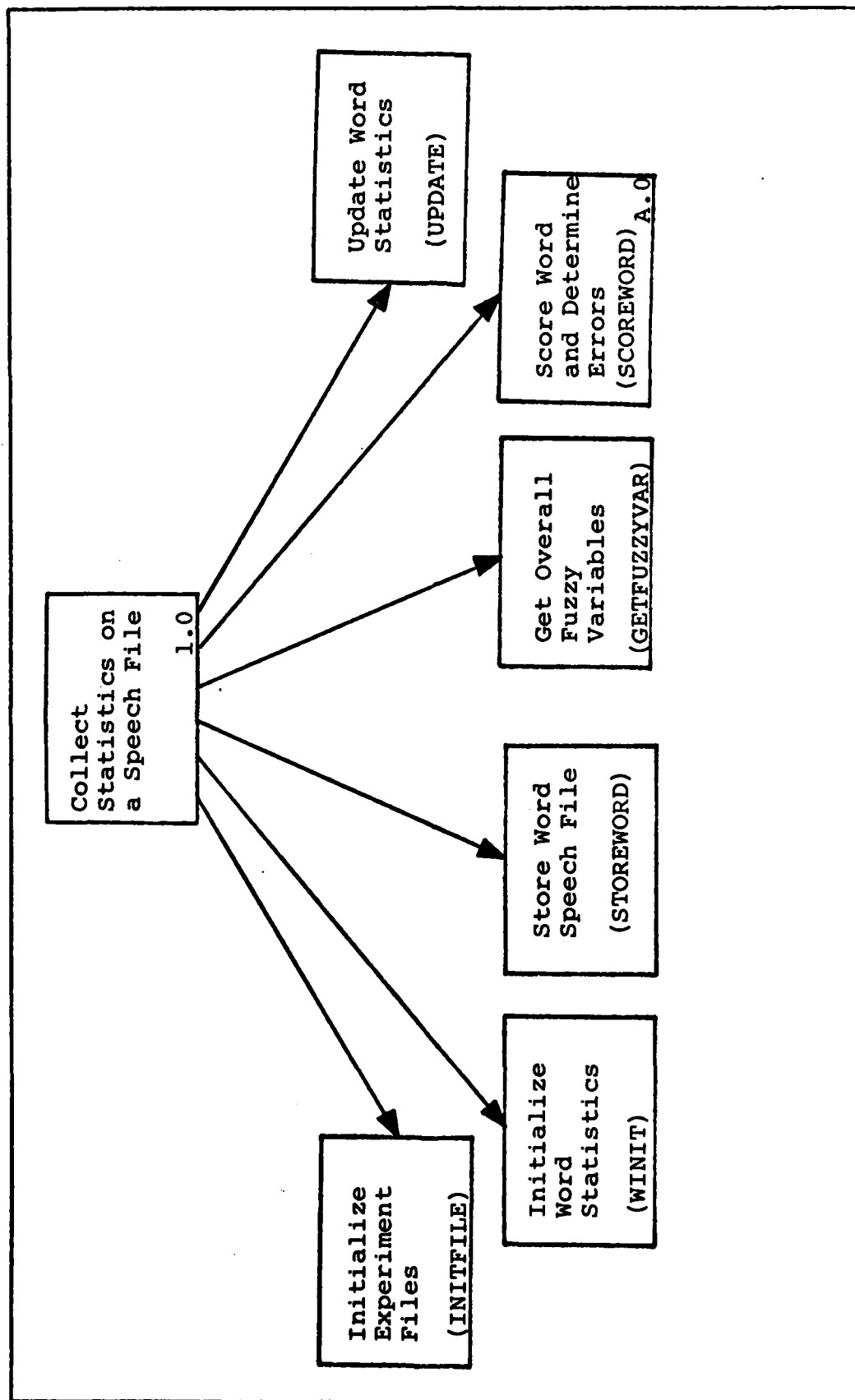


Fig 28. Structure Chart 1.0: Collect Statistics on a Speech File

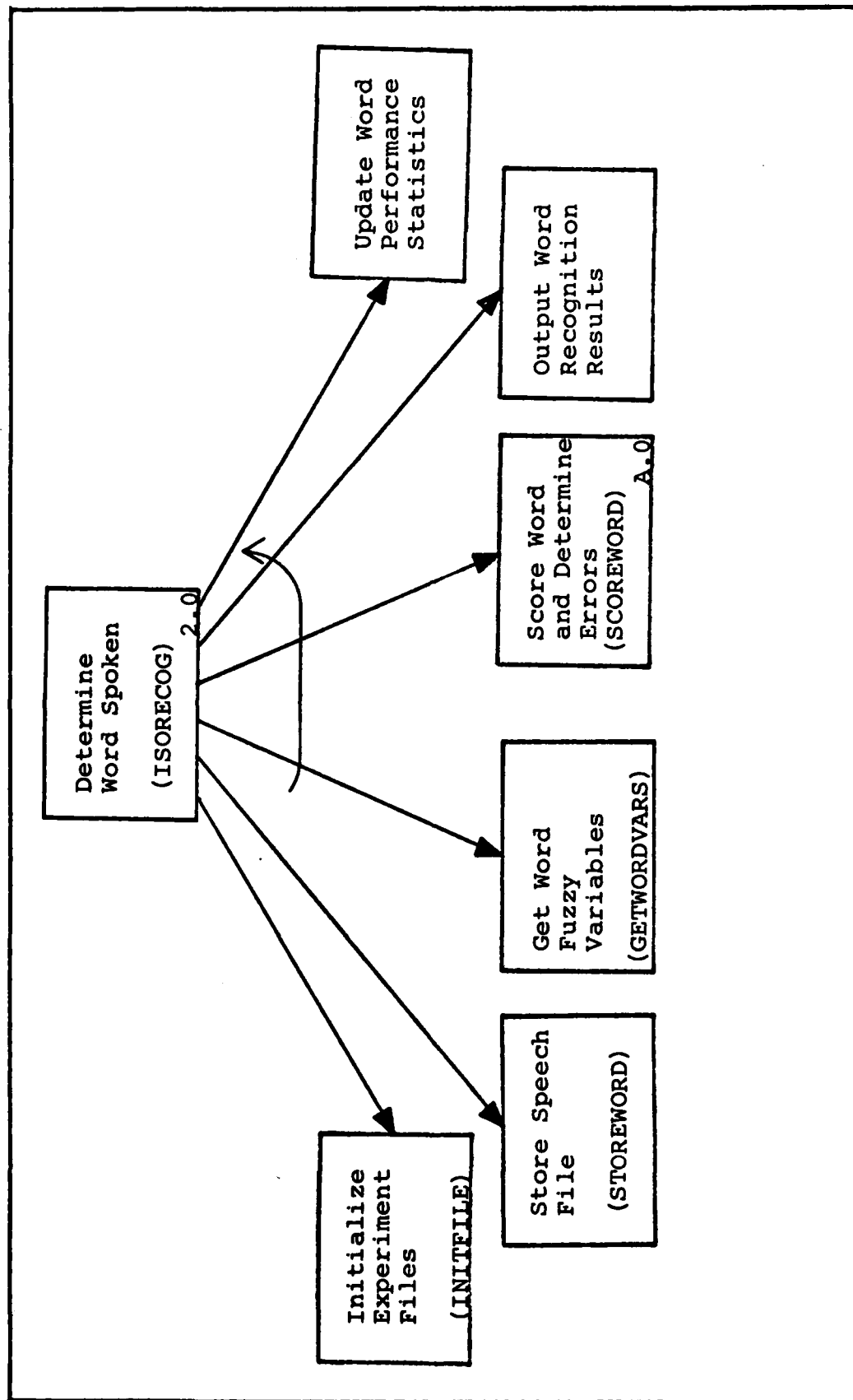


Fig. 29. Structure Chart 2.0: Determine Word Spoken.
Procedure "ISORECOG"

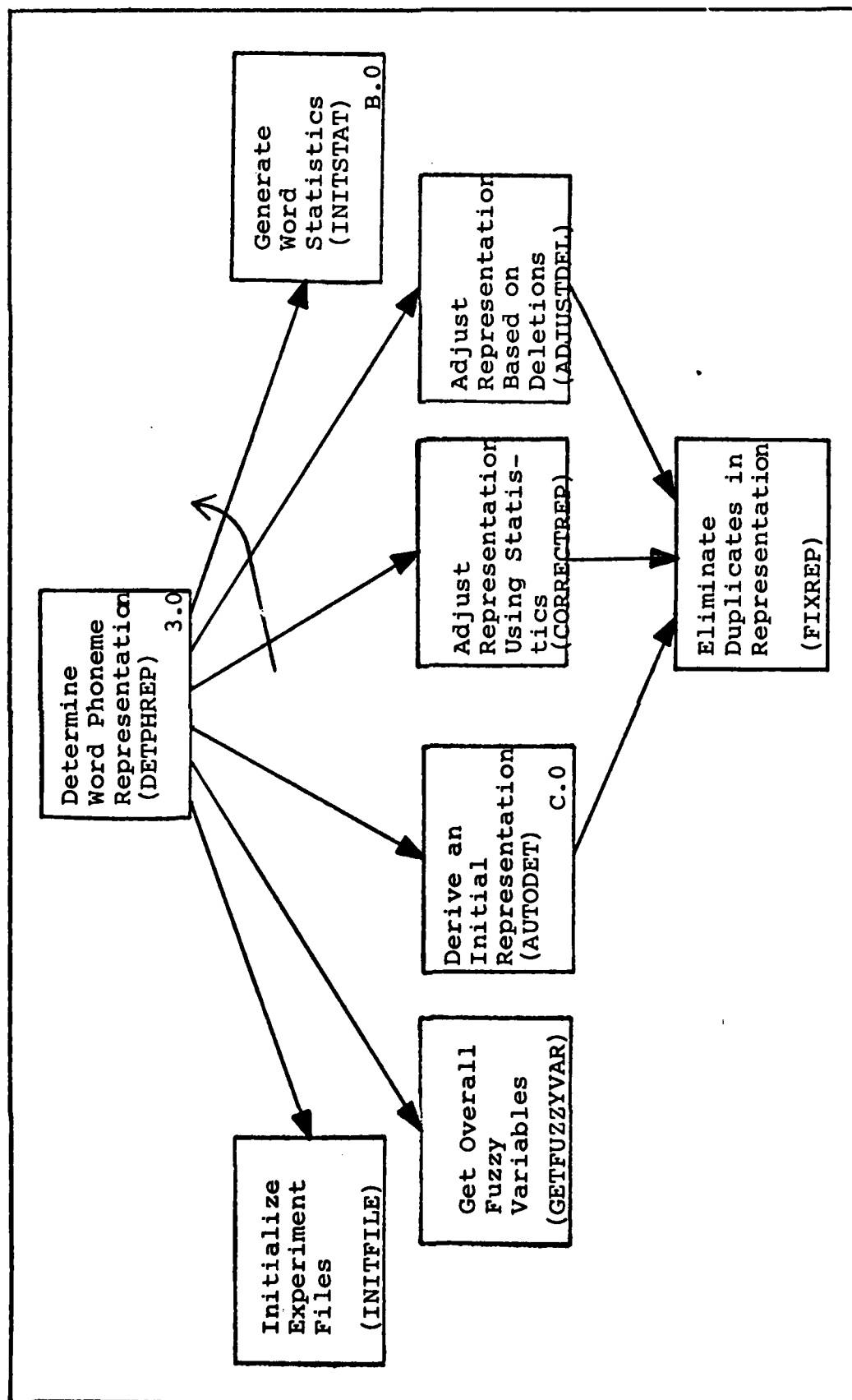


Fig. 30. Structure Chart 3.0: Determine Word Phoneme Representation.
Procedure "DETPHREP"

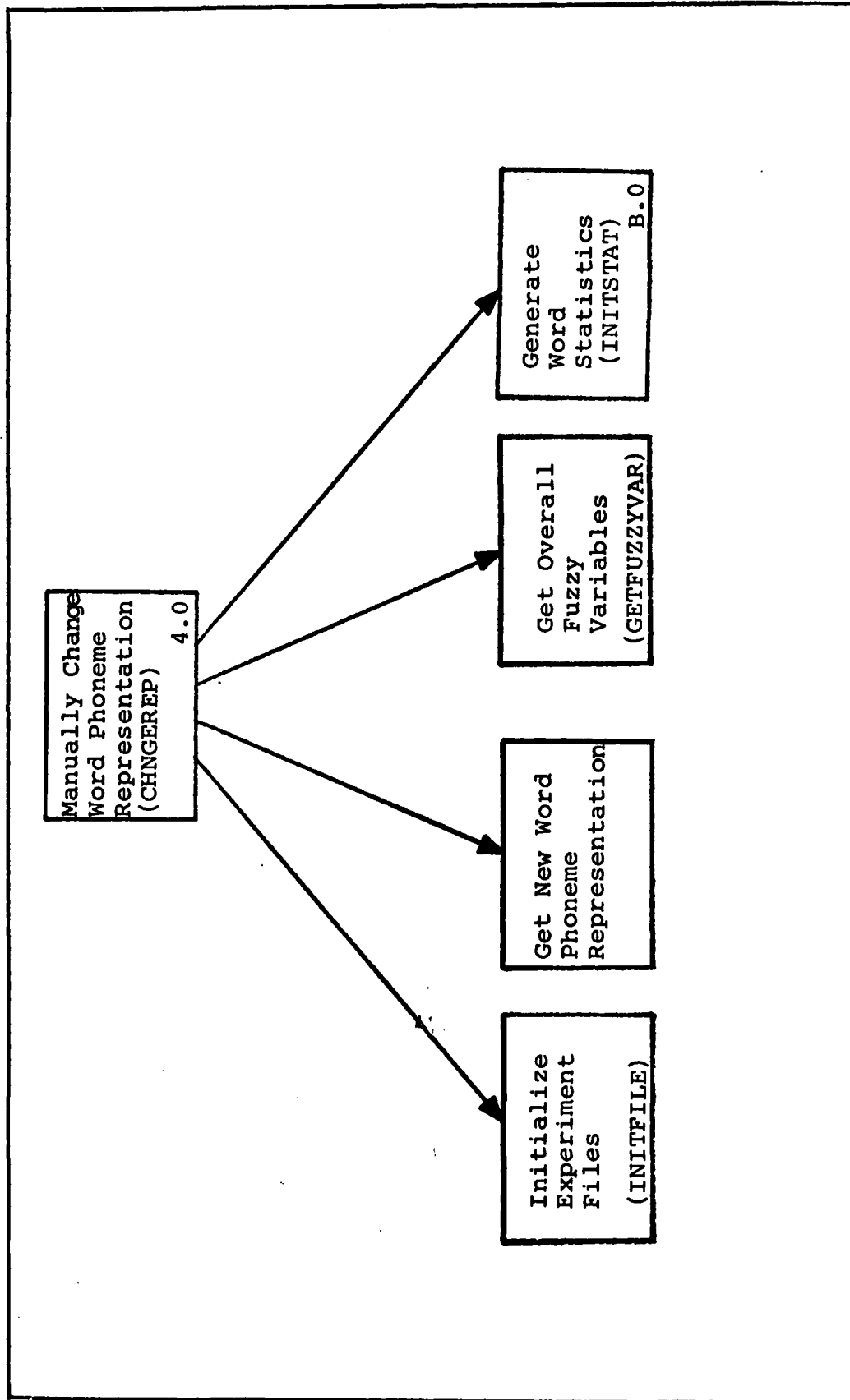


Fig. 31. Structure Chart 4.0: Manually Change Word Phoneme Representation.
Procedure "CHNGEREP"

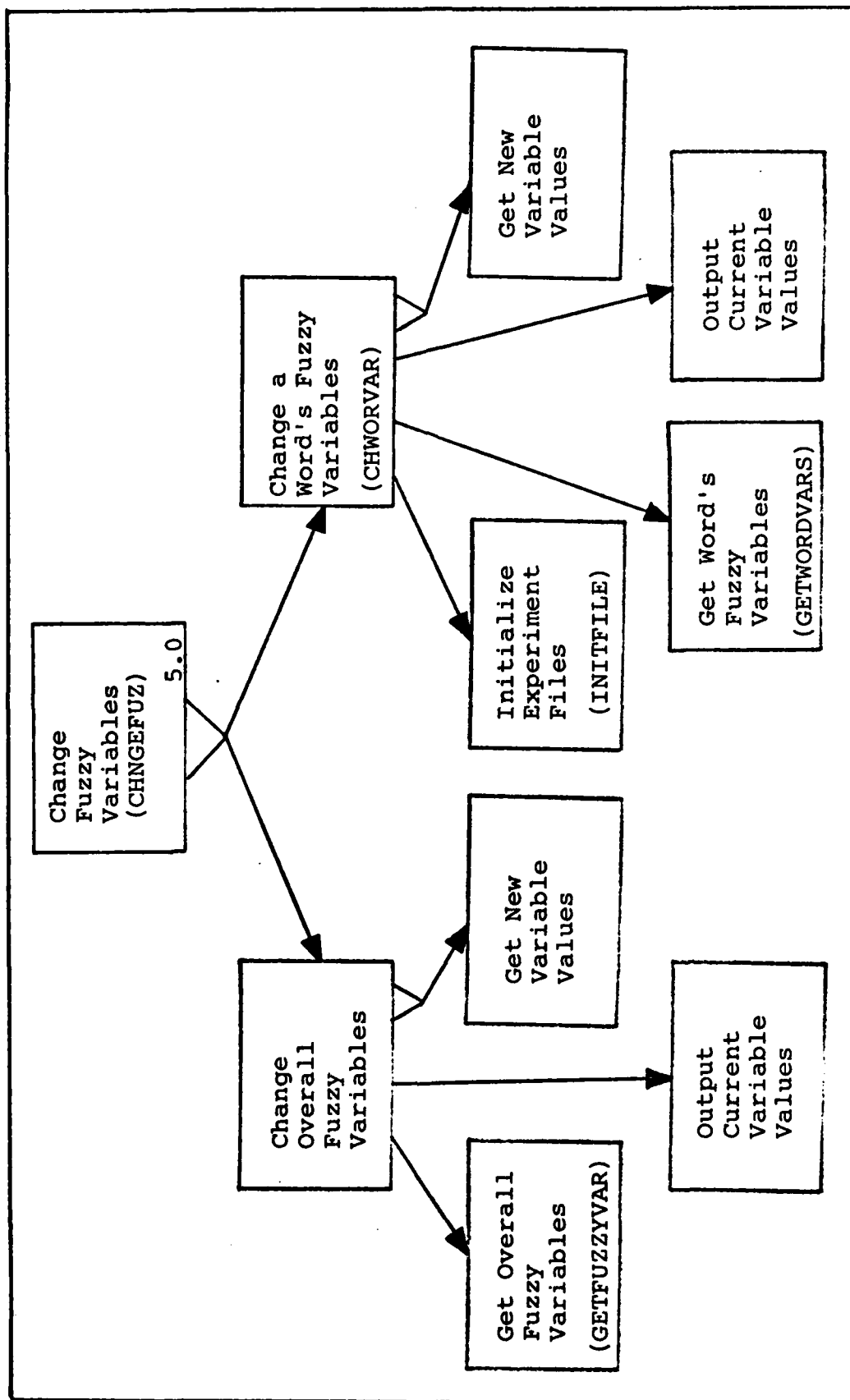


Fig. 32. Structure Chart 5.0: Change Fuzzy Variables.
Procedure "CHNGEFUZ"

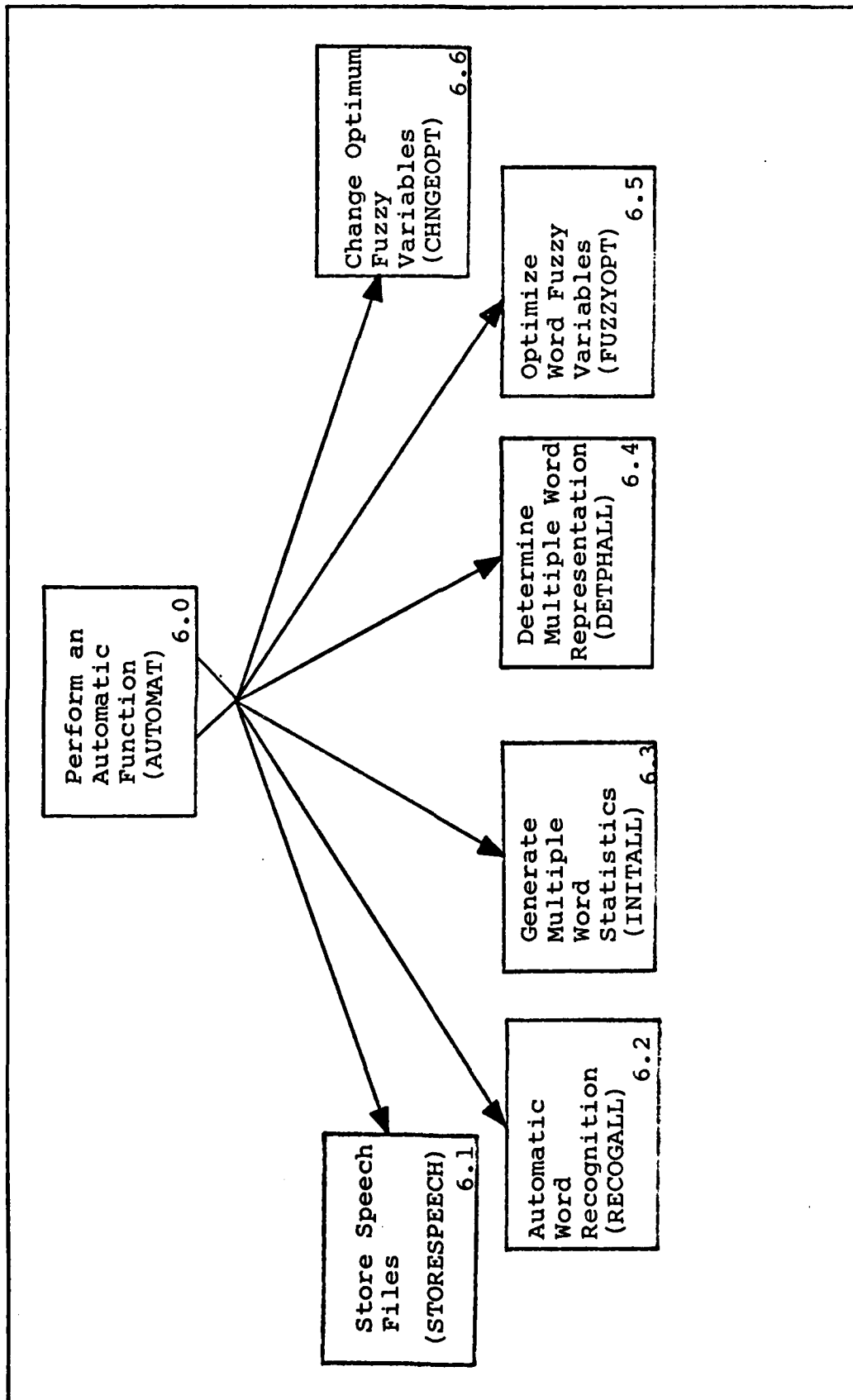


Fig. 33. Structure Chart 6.0: Run an Automated Procedure.
Procedure "AUTOMAT"

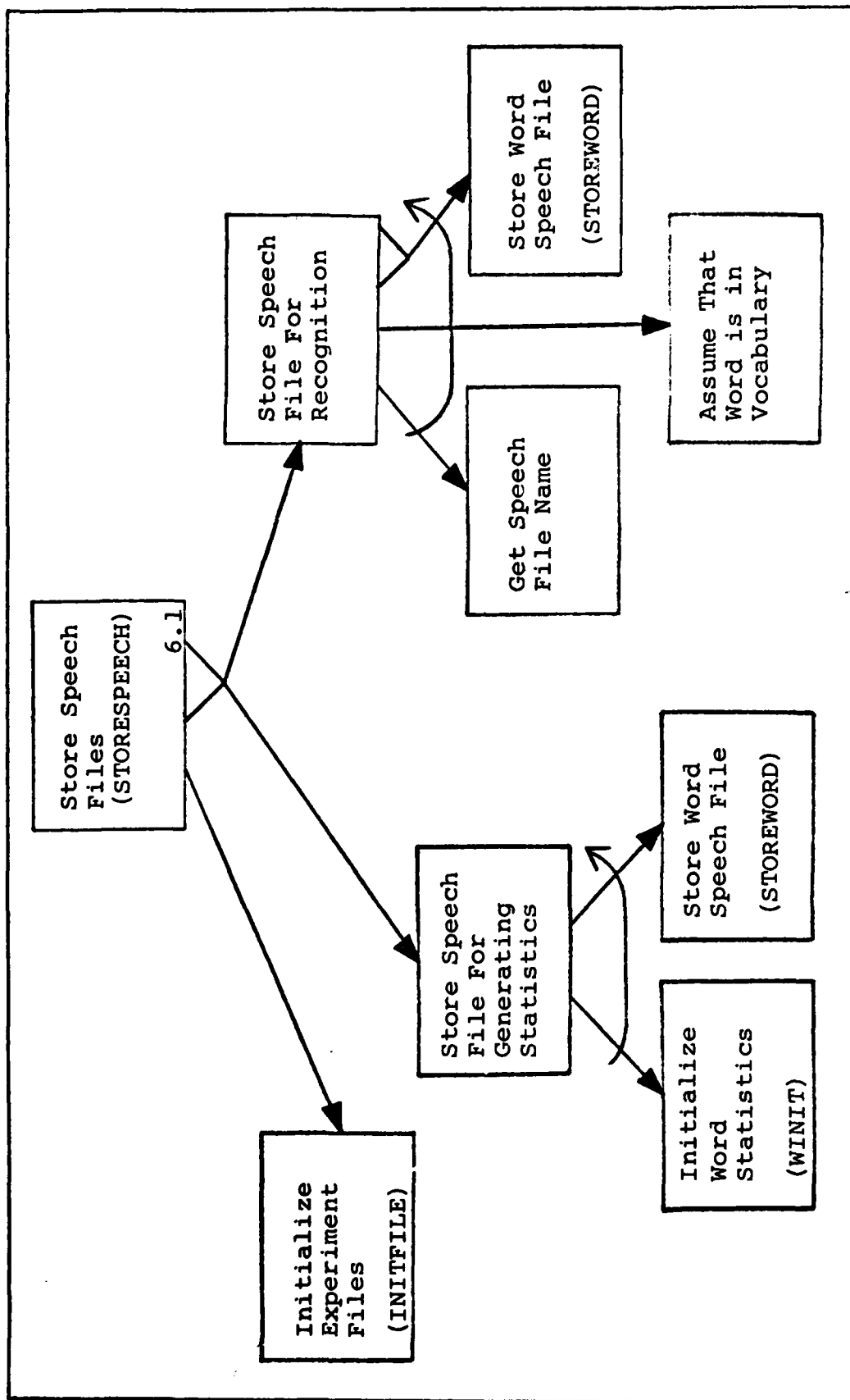


Fig. 34. Structure Chart 6.1: Store Speech Files.
Procedure "STORESPEECH"

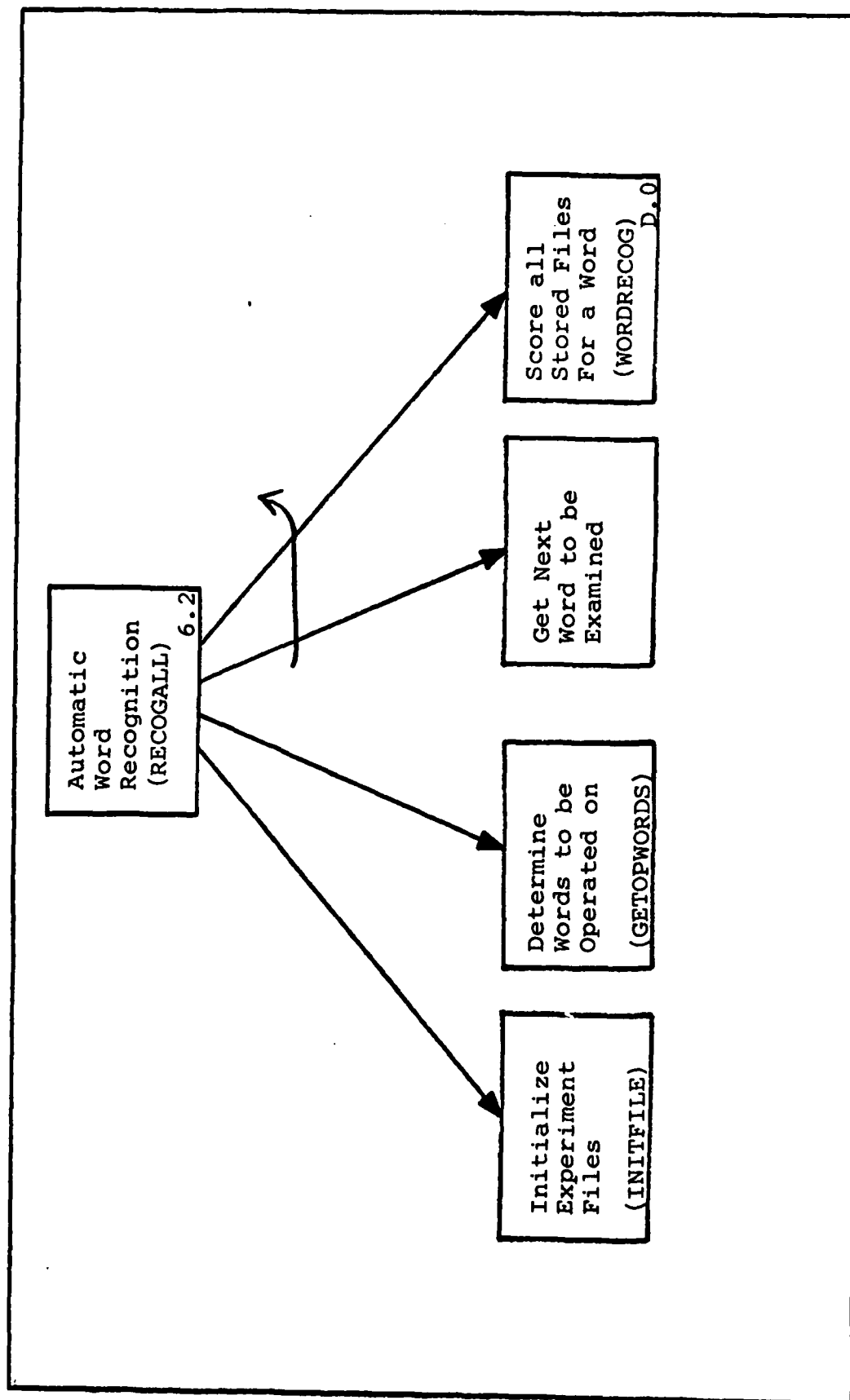


Fig. 35. Structure Chart 6.2: Automatic Word Recognition.
Procedure "RECOGALL"

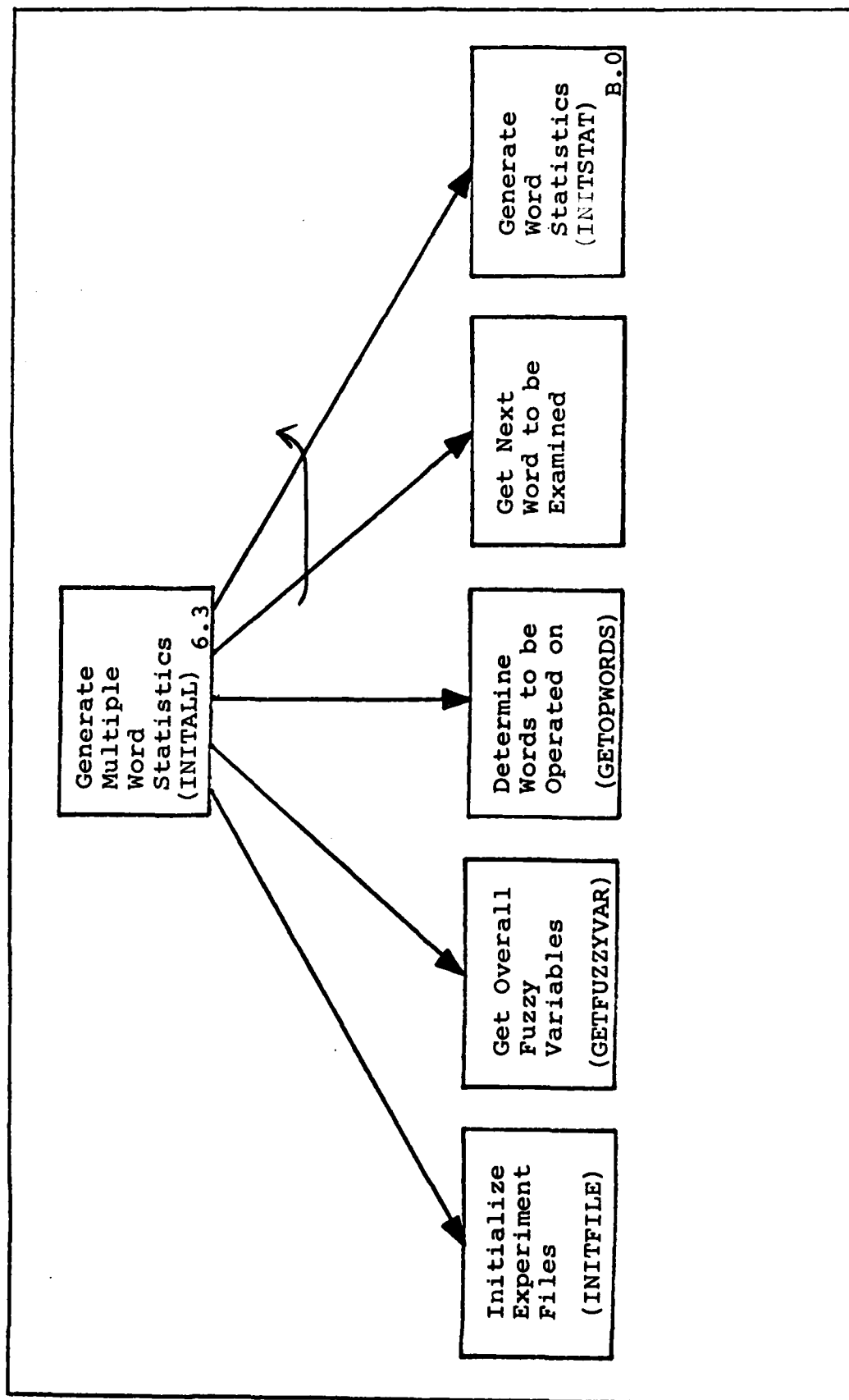


Fig. 36. Structure Chart 6.3: Generate Multiple Word Statistics.
Procedure "INITALL"

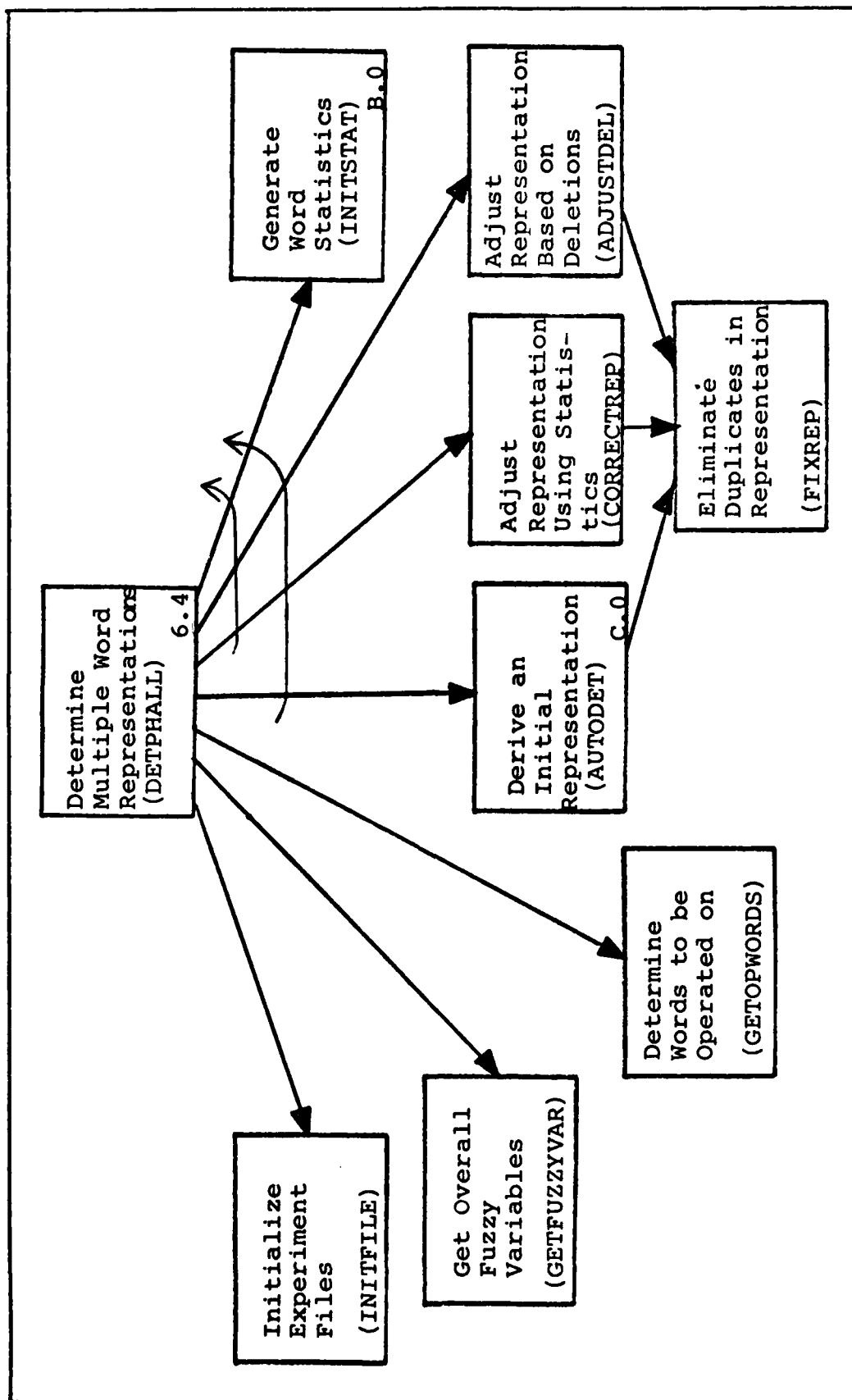


Fig. 37. Structure Chart 6.4: Determine Multiple Word Representations. Procedure "DETPHALL"

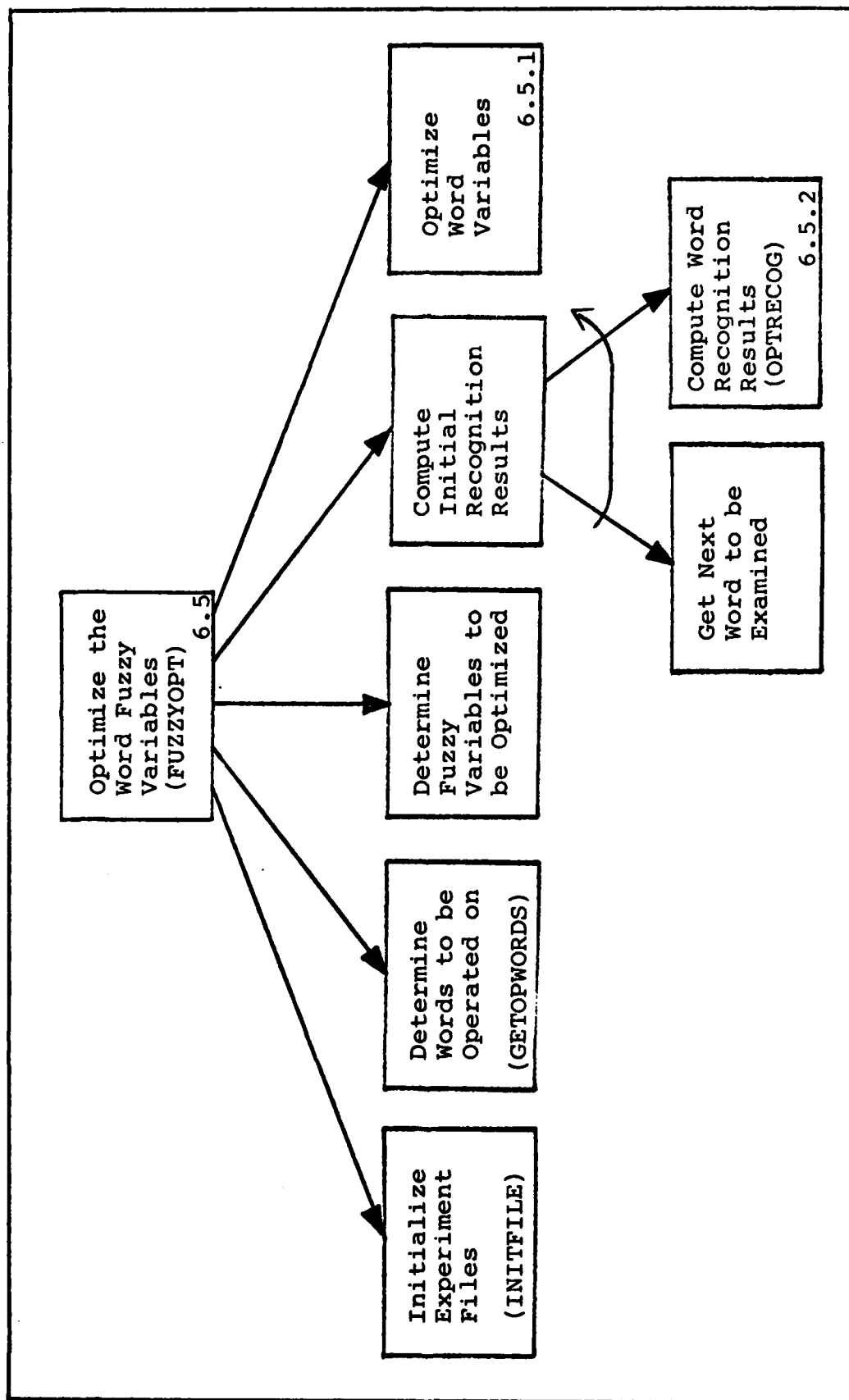


Fig. 38. Structure Chart 6.5: Optimize the Word Fuzzy Variables. Procedure "FUZZYOPT"

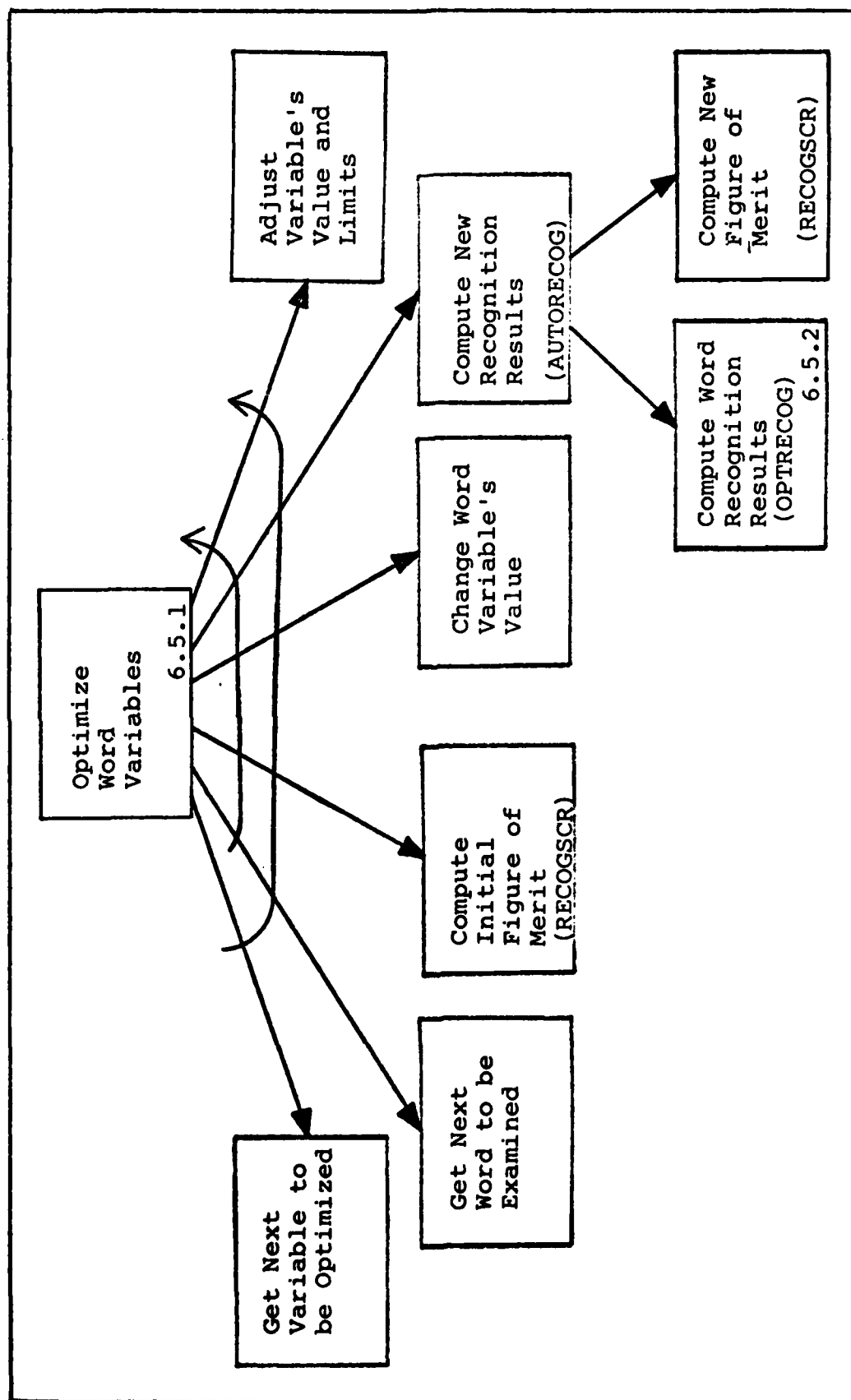


Fig. 39. Structure Chart 6.5.1: Optimize Word Variables

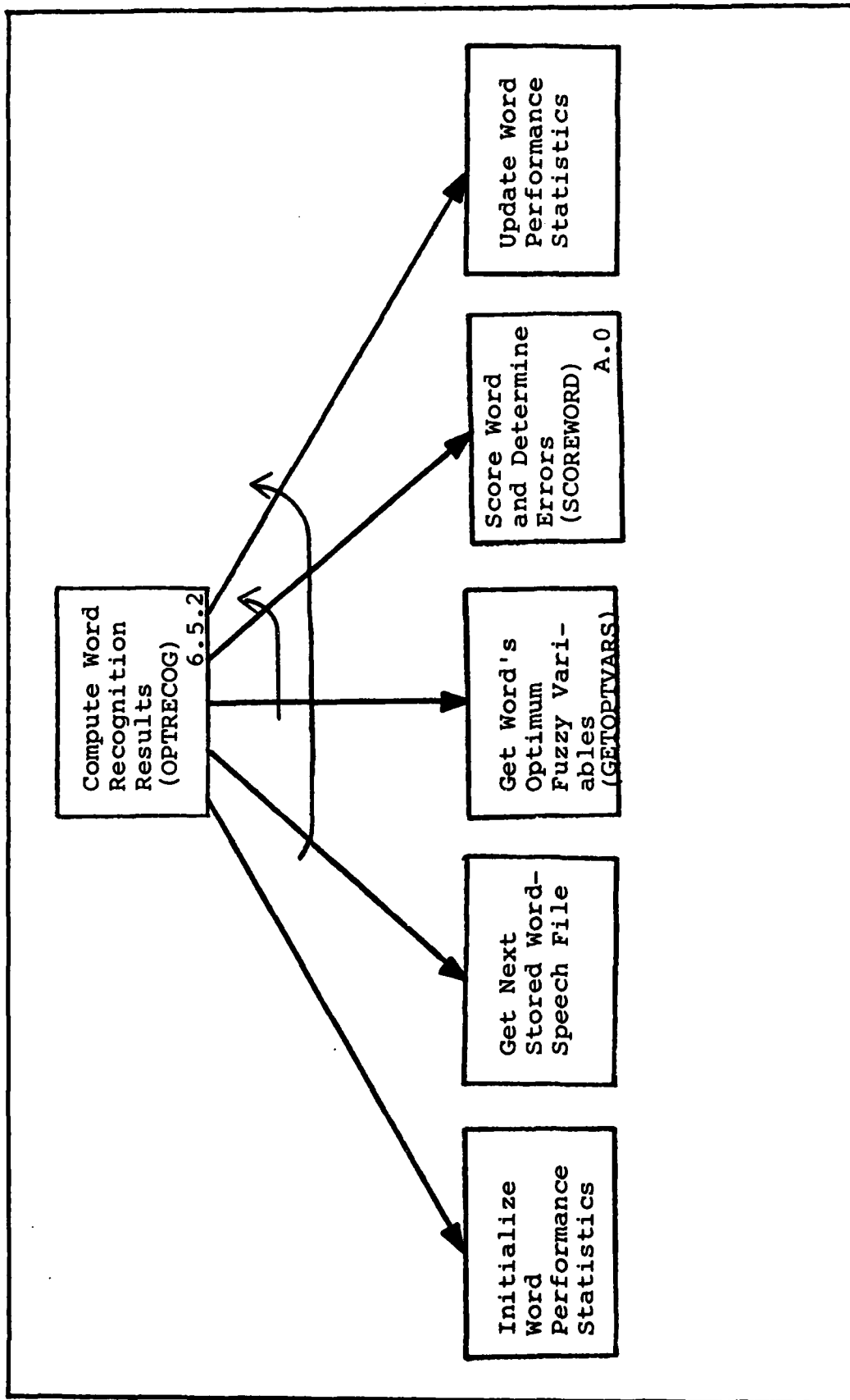


Fig. 40. Structure Chart 6.5.2: Compute Word Recognition Results.
Procedure "OPTRECOG"

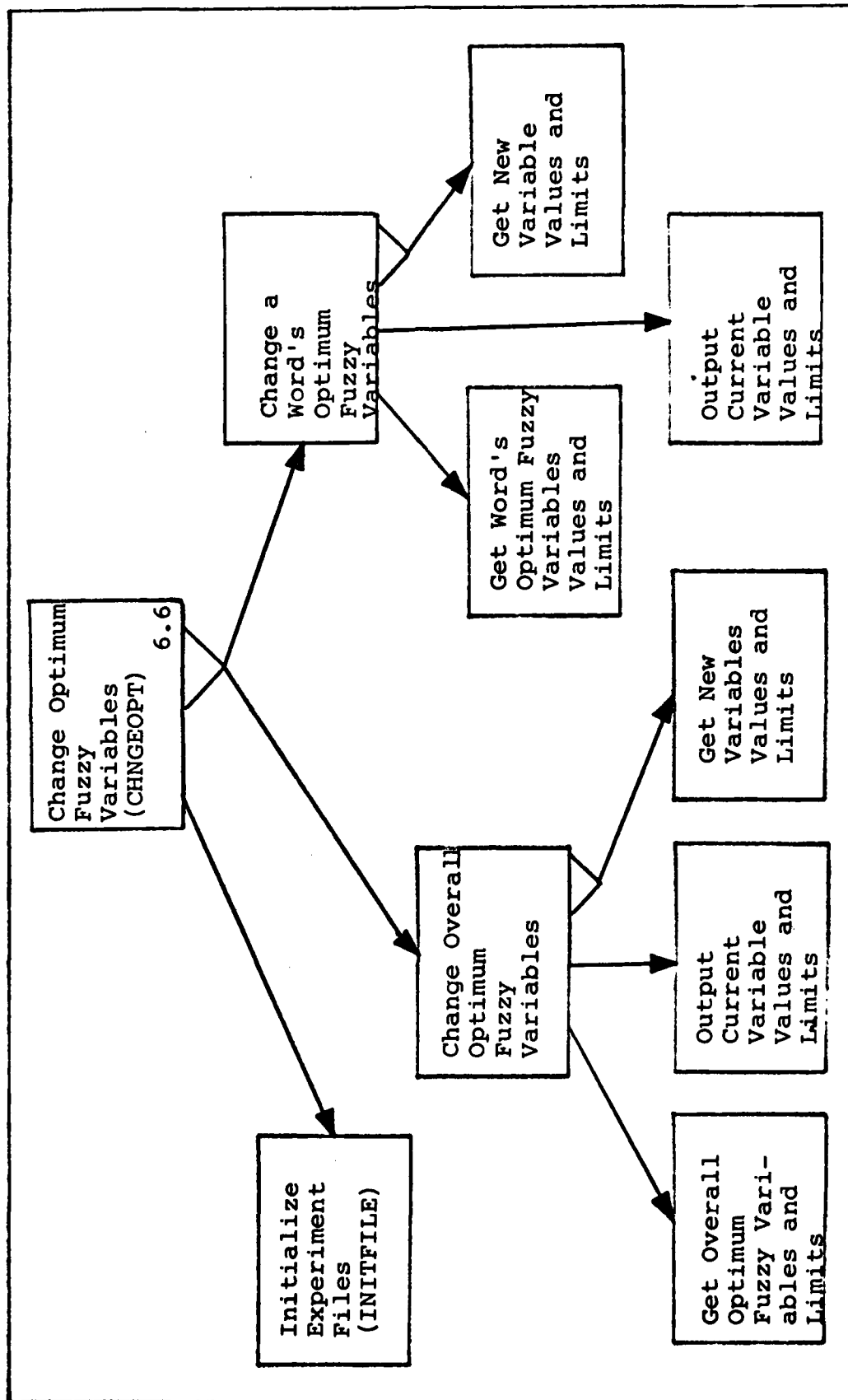


Fig. 41. Structure Chart 6.6: Change Optimum Fuzzy Variables.
Procedure "CHNGEOPT"

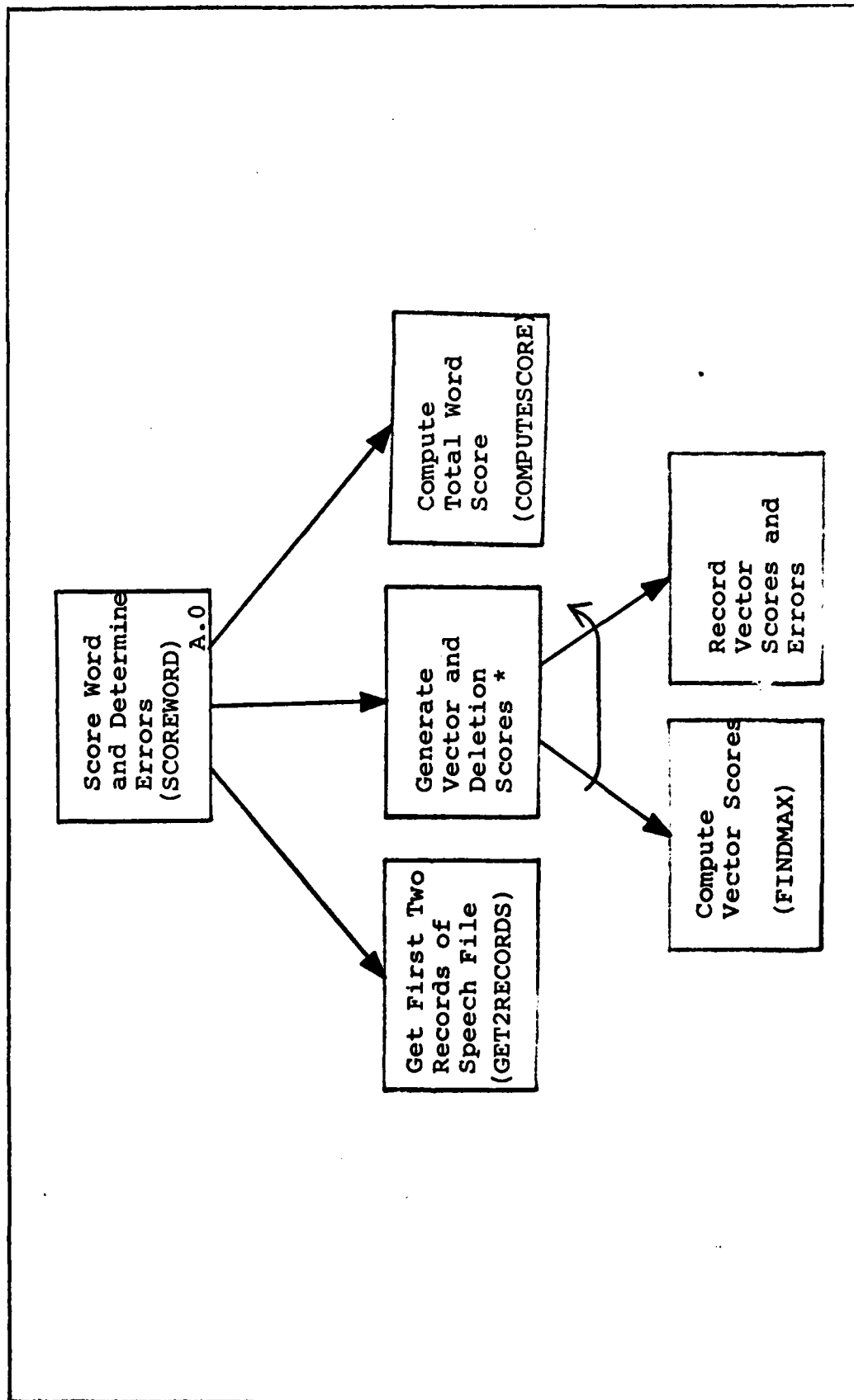


Fig. 42. Structure Chart A.0: Score Word and Determine Errors.
Procedure "SCOREWORD"

NOTE * - This function varies depending on the number of word phonemes and vectors remaining. Refer to procedures WGE2SGE3, WGE2SLE2, WEQ1SGE3, WEQ1SLE2.

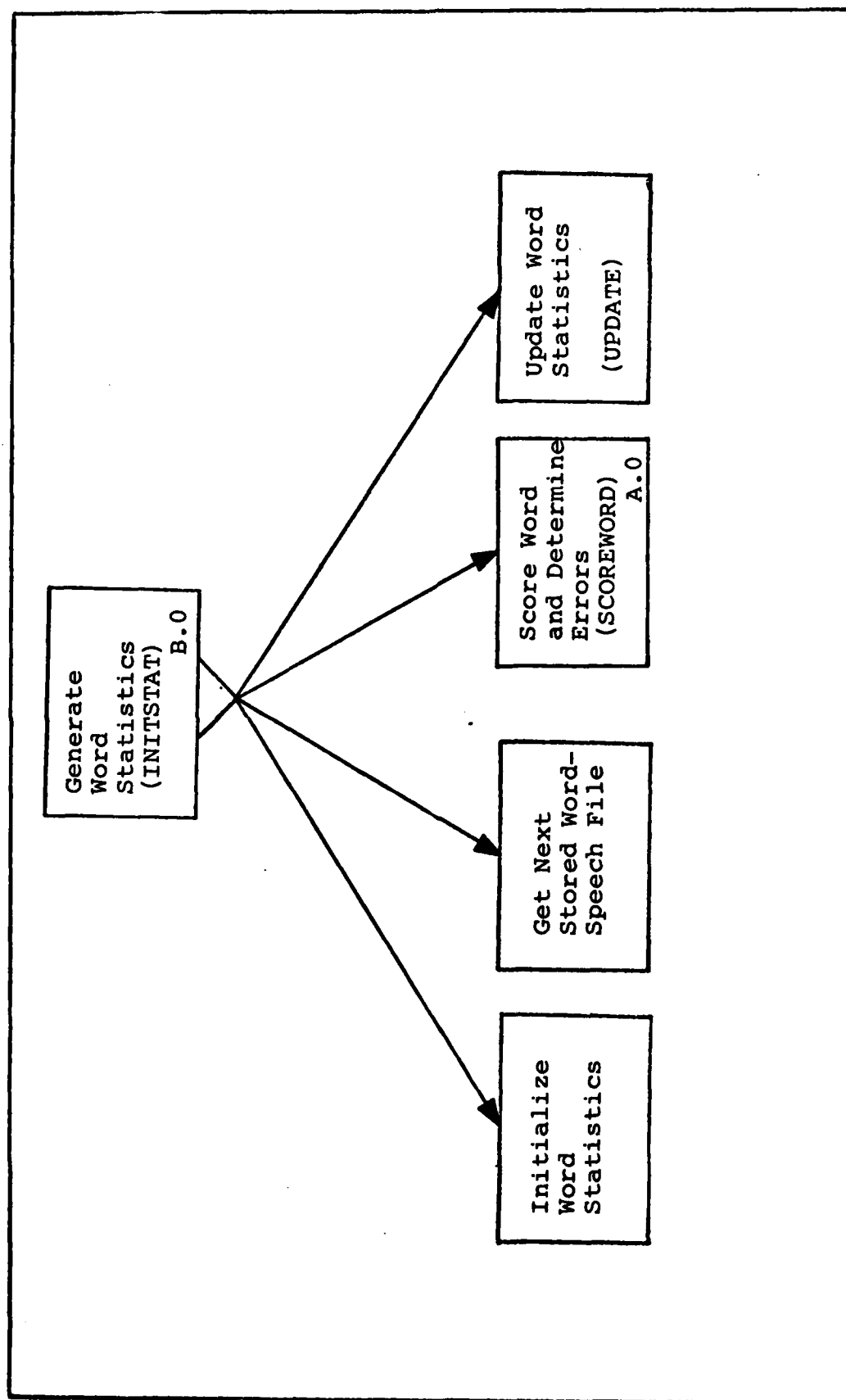


Fig. 43. Structure Chart B.0: Generate Word Statistics.
Procedure "INITSTAT"

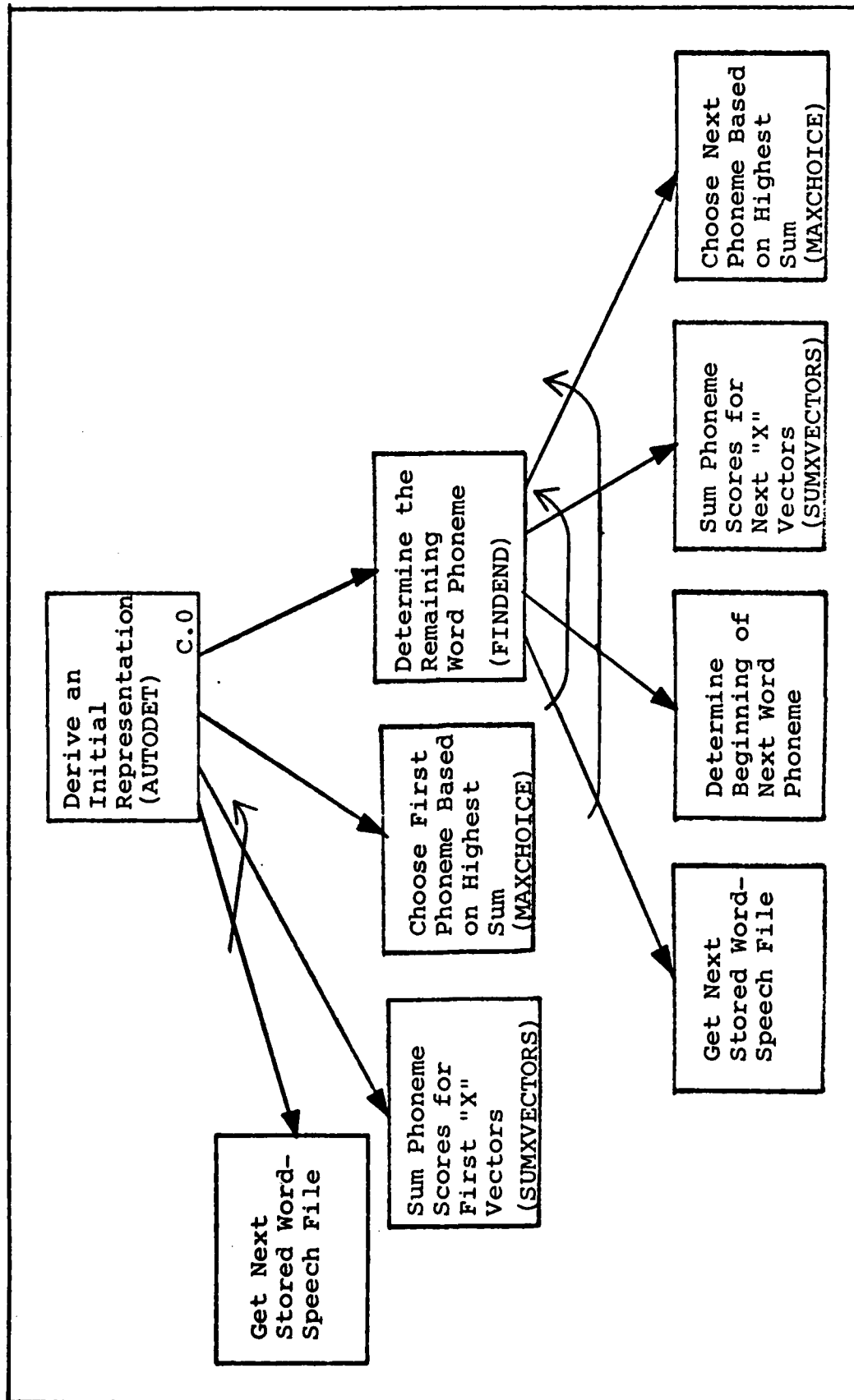


Fig. 44. Structure Chart C.0: Derive an Initial Representation.
Procedure "AUTODET"

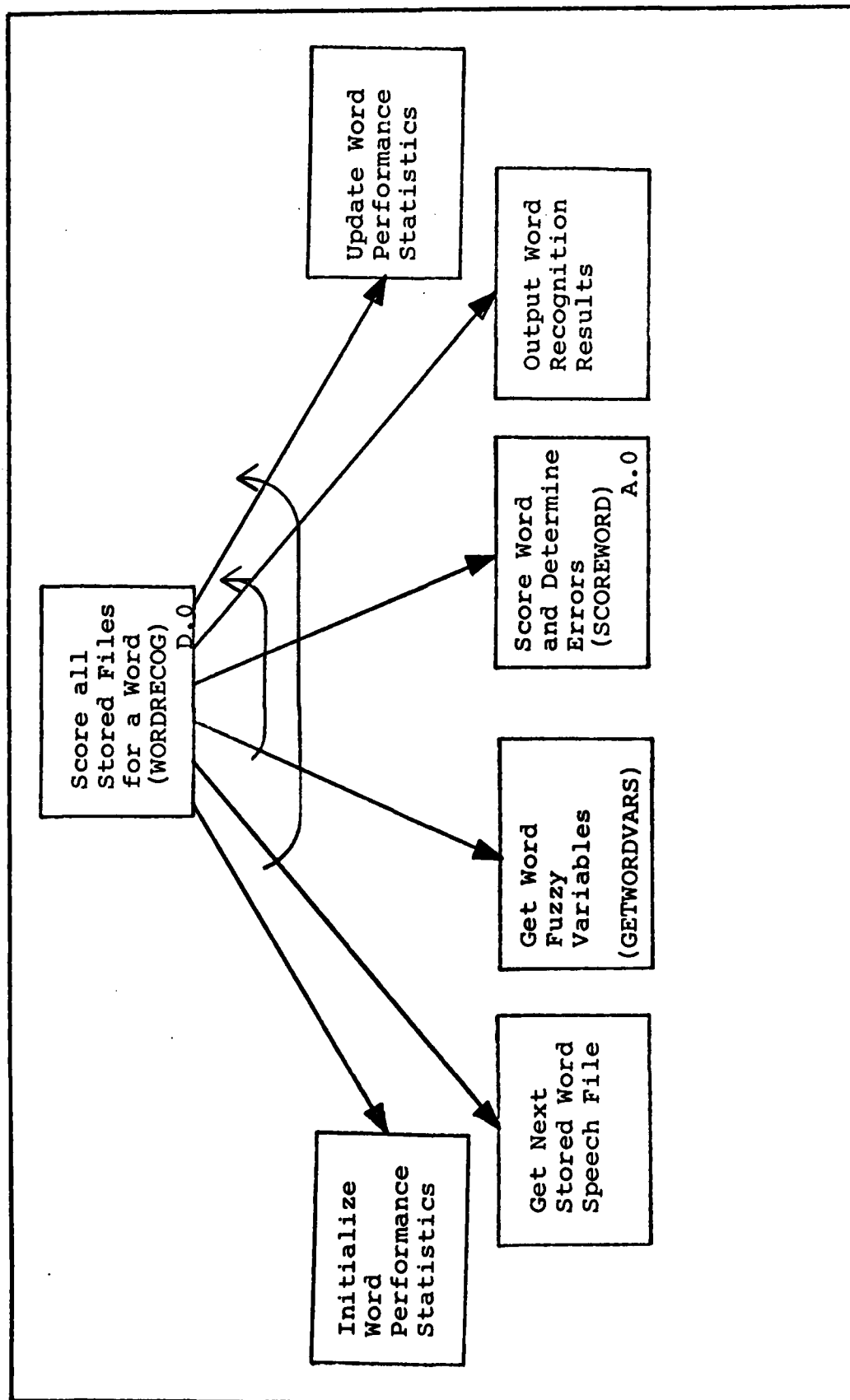


Fig. 45. Structure Chart D.0: Score All Stored Files for a Word.
Procedure "WORDRECOG"

APPENDIX D
Computer Programs

(*****)

PROGRAM LEARNWORD DOCUMENTATION

WRITTEN BY: 2LT GERARD J. MONTGOMERY

DATE: 15 JUNE 1982

GENERAL PROGRAM DESCRIPTION

THIS PROGRAM PROVIDES THE CAPABILITY AND SUPPORT NECESSARY TO PERFORM ISOLATED WORD RECOGNITION GIVEN THE OUTPUT OF AN ACOUSTIC PROCESSOR. THE ONLY FUNCTION THAT IS NOT PROVIDED IN THIS PROGRAM, IS A FUNCTION TO OUTPUT THE STATISTICS AND RECOGNITION RESULTS COLLECTED IN FILES SFILE AND WFILE. THIS FUNCTION IS PROVIDED BY PROGRAM OUTSTAT. THE MAIN FUNCTIONS THAT PROGRAM LEARNWORD PROVIDES INCLUDE:

- 1A. COLLECTING STATISTICS ON A GIVEN SPEECH FILE (TO BE USED FOR TRAINING).
- 1B. ADDING A NEW WORD TO THE VOCABULARY, AND INITIALIZING THE NECESSARY DATA STRUCTURES FOR THE WORD.
2. RECOGNIZING THE WORD SPOKEN IN A GIVEN SPEECH FILE.
3. DETERMINING A WORD'S PHONEME REPRESENTATION FROM THE SPEECH FILES USED FOR TRAINING.
4. CHANGING THE PHONEME REPRESENTATION OF A WORD.
5. CHANGING THE VALUES OF THE OVERALL OR WORD FUZZY VARIABLES.
6. RUNNING AN AUTOMATED PROCEDURE (REFER TO THE DESCRIPTION FOR PROCEDURE AUTOMAT).

NOTE: A DETAILED PRESENTATION OF THE PROGRAM'S STRUCTURE IS PROVIDED BY THE STRUCTURE CHARTS SHOWN IN APPENDIX 1.

MAIN DATA STRUCTURES

INPUT/OUTPUT DATA FILES:

OUT2 - TEXT FILE USED FOR SPEECH FILE INPUT.

RESULTS - OUTPUT TEXT FILE FOR STORING RECOGNITION RESULTS.

STATSOUT - OUTPUT TEXT FILE FOR OBTAINING WORD STATISTICS, AND OTHER OUTPUT DATA.

DATA FILES:

WFILE* - A RANDOM-ACCESS FILE CONSISTING OF ONE RECORD PER WORD IN THE VOCABULARY. THE PURPOSE OF THIS FILE IS TO MAINTAIN GENERAL INFORMATION FOR EACH WORD IN THE VOCABULARY. EACH RECORD CONTAINS THE FOLLOWING DATA:

VARIABLE -----	DESCRIPTION -----
SPELLING	- WORD SPELLING
PHREP	- ARRAY CONTAINING WORD'S PHONEME REPRESENTATION.
WSTHR,WSUBE,WSURF,WINSE,	
WINSF,WDELE,WDELF,	
WDELG,WDCNE,WDCNF,	
WDCNG,WSFE,WSFF,	
WCHVE,WCHVF,	
WSTATE,WSTATF,WSTATG,	
WTHR1E,WTHR1F,WTHR2E,	
WTHR2F	- WORD'S FUZZY VARIABLES (SEE THE OVERALL FUZZY VARIABLES DESCRIPTIONS FOR AN EXPLANATION OF A SPECIFIC VARIABLE).

SFILE* - A RANDOM-ACCESS FILE CONSISTING OF ONE RECORD PER WORD IN THE VOCABULARY. THE PURPOSE OF THIS FILE IS TO MAINTAIN STATISTICAL DATA FOR EACH WORD IN THE VOCABULARY. EACH RECORD CONTAINS THE FOLLOWING DATA:

VARIABLE -----	DESCRIPTION -----
NOSPOKE	- NUMBER OF TRAINING FILES USED TO OBTAIN THE WORD STATISTICS.
NOATTEMPT	- NUMBER OF RECOGNITION ATTEMPTS MADE FOR THE WORD.
NOCORRECT	- NUMBER OF TIMES THE WORD WAS CORRECTLY RECOGNIZED.
NDEL	- ARRAY CONTAINING THE NUMBER OF TIMES EACH WORD PHONEME WAS DELETED IN THE TRAINING SET.
XSUBY	- ARRAY CONTAINING A SCORE BASED ON THE NUMBER OF TIMES EACH PHONEME WAS SUBSTITUTED FOR EACH WORD PHONEME.
XINAFIZ	- ARRAY CONTAINING A SCORE BASED ON THE NUMBER OF TIMES EACH PHONEME WAS INSERTED FOR EACH WORD PHONEME.
TOTSCORE	- TOTAL RECOGNITION SCORE.
TOTDIFF	- TOTAL DIFFERENCE BETWEEN THE WORD'S RECOGNITION SCORE AND THE NEXT HIGHEST SCORE. (ONLY CALCULATED WHEN WORD WAS CORRECTLY RECOGNIZED)
MINDIFF	- THE MINIMUM DIFFERENCE BETWEEN THE WORD'S RECOGNITION SCORE AND THE NEXT HIGHEST SCORE.

(ONLY CALCULATED WHEN WORD WAS CORRECTLY
RECOGNIZED)

SPECH* - A RANDOM-ACCESS FILE CONSISTING OF ONE OR MORE RECORDS
PER SPEECH FILE. THE PURPOSE OF THIS FILE IS TO
MAINTAIN A NUMBER OF SPEECH FILES THAT CAN BE EASILY
ACCESSED BY THE PROGRAM TO INITIALIZE WORD STATISTICS ,
AND TO DETERMINE A WORD'S PHONEME REPRESENTATION.
EACH RECORD CONTAINS THE FOLLOWING DATA:

VARIABLE	DESCRIPTION
-----	-----
SPELL	- EXACT SPELLING OF THE WORD SPOKEN.
CONTFLAG	- CONTINUATION FLAG INDICATING WHETHER OR NOT THE NEXT RECORD IS A CONTINUATION OF THE CURRENT SPEECH FILE.
SPKR	- THE SPEAKER OF THE WORD.
ICHPHONE	- ARRAY CONTAINING THE TOP PHONEME GUESSES FOR EACH SEGMENT OF SPEECH.
ICHVALUE	- ARRAY CONTAINING A RELATIVE WEIGHT FOR EACH PHONEME GUESS.
ISFACTOR	- ARRAY CONTAINING A RELATIVE WEIGHT FOR EACH VECTOR OF SPEECH INDICATING THE LEVEL OF CONFIDENCE THE ACOUSTIC PROCESSOR HAD IN CHOOSING THE GIVEN PHONEMES FOR A SEGMENT OF SPEECH.
IPHONE	- ARRAY USED BY THE RECOGNITION ROUTINE TO INDICATE THE WORD PHONEME CHOSEN FOR EACH VECTOR IN THE SPEECH FILE.
ISCORE	- ARRAY USED BY THE RECOGNITION ROUTINE TO INDICATE A SCORE FOR THE WORD PHONEME CHOSEN FOR EACH SPEECH VECTOR.
IVERROR	- ARRAY USED BY THE RECOGNITION ROUTINE TO INDICATE THE ERROR THAT OCCURRED FOR EACH SPEECH VECTOR.

RECOG* - IDENTICAL TO SPECH, EXCEPT THE PURPOSE OF THIS FILE IS TO
MAINTAIN A NUMBER OF SPEECH FILES THAT CAN BE EASILY
ACCESSED BY THE PROGRAM FOR OBTAINING RECOGNITION DATA.

FUZZYVAR - A TEXT FILE CONTAINING THE OVERALL FUZZY VARIABLES THAT
ARE USED FOR CALCULATING THE WORD STATISTICS.

OPTFUZZY - A TEXT FILE CONTAINING INITIAL FUZZY VARIABLE VALUES, MINIMUM
LIMITS, AND MAXIMUM LIMITS. THIS FILE IS USED BY PROCEDURE
FUZZYOPT TO INITIALIZE THE OPTIMUM FUZZY VARIABLES OF WORDS
ADDED TO THE VOCABULARY.

FZOPT* - A RANDOM-ACCESS FILE CONSISTING OF ONE RECORD PER WORD. THE PURPOSE OF THIS FILE IS TO MAINTAIN EACH WORD'S OPTIMUM FUZZY VARIABLES AND VARIABLE LIMITS. THIS FILE IS USED BY PROCEDURE FUZZYOPT.

NOTE:

* - INDICATES THAT THE ACTUAL FILE NAME IS PRECEDED BY THREE INITIALS. FOR EXAMPLE, THE FOLLOWING FILES ARE CURRENTLY BEING USED TO COLLECT DATA FOR MULTIPLE SPEAKERS:

ALLWFILE
ALLSFILE
ALLSPECH
ALLRECOG
ALLFZOPT

GLOBAL CONSTANTS

CONSTANT	DESCRIPTION
SPELLENGTH	- THE MAXIMUM NUMBER OF LETTERS FOR SPELLING A WORD.
MAXPHONES	- THE MAXIMUM NUMBER OF PHONEMES IN A WORD'S PHONEME REPRESENTATION.
MAXPHPLUS1	- MAXPHONES PLUS 1.
NOPHONEMES	- THE NUMBER OF PROTOTYPE PHONEMES.
NOGUESSES	- THE NUMBER OF GUESSES(CHOICES) PROVIDED BY THE ACOUSTIC PROCESSOR.
MAXNOWORDS	- THE MAXIMUM NUMBER OF WORDS IN THE VOCABULARY.
RCRDLENGTH	- THE NUMBER OF VECTORS PER RECORD USED IN STORING SPEECH FILES.
MAXVECTORS	- THE MAXIMUM NUMBER OF VECTORS OF A SPEECH FILE TO BE IN MEMORY AT ONE TIME (2 * RCRDLENGTH).
NOTOAVG	- THE NUMBER OF VECTORS TO BE AVERAGED BY THE RECOGNITION ROUTINE TO DETERMINE IF A PHONEME IS A TRANSITION BETWEEN WORD PHONEMES.
NOFUZZYVARS	- TOTAL NUMBER OF FUZZY VARIABLES USED IN THE PROGRAM.
BEGINWORD	- THE STARTING VECTOR OF A WORD IN A SPEECH FILE. (PROVIDED FOR THE FUTURE IMPLEMENTATION OF A CONTINUOUS SPEECH ALGORITHM)

GLOBAL VARIABLES

VARIABLES	DESCRIPTION
WDATA	- ONE RECORD OF DATA STORED IN FILE WFILE.
WSTATS	- ONE RECORD OF DATA STORED IN FILE SFILE.
IOUT	- ONE RECORD OF DATA STORED IN FILES SPECH OR RECOG.
FLIMIT	- ONE RECORD OF DATA STORED IN FILE FZOPT.
OPTION	- VARIABLE USED TO INDICATE THE FUNCTION TO BE PERFORMED.
STOPFLAG	- FLAG USED TO STOP PROGRAM.
CHNGEFLAG	- FLAG USED BY PROCEDURE AUTODET TO DETERMINE WHEN TO STOP.
WORDSCORE	- WORD SCORE OBTAINED BY PROCEDURE SCOREWORD.
DELCNT	- NUMBER OF DELETIONS OBTAINED BY PROCEDURE SCOREWORD.
DELW	- TOTAL DELETION SCORE OBTAINED BY PROCEDURE SCOREWORD.
SCORECOUNT	- TOTAL NUMBER OF VECTORS IN A SPEECH FILE OBTAINED BY PROCEDURE SCOREWORD.
INSRTSCORE	- TOTAL INSERTION SCORE OBTAINED IN PROCEDURE SCOREWORD.
PWORD	- TOTAL WORD SCORE EXCLUDING THE EFFECTS OF DELETIONS OBTAINED BY PROCEDURE SCOREWORD.
WORD	- WORD CURRENTLY BEING OPERATED ON.
BEGINREC	- BEGINNING RECORD OF A SPEECHFILE.
IRECORD	- CURRENT RECORD BEING OPERATED ON.
ENDFLAG	- FLAG USED TO INDICATE WHEN A PROCESS HAS BEEN COMPLETED.
SPEAKER	- THE THREE INITIALS PRECEDING THE DATA FILES.
FILENAME	- FILE TITLE OF FILE TO BE USED.
ENDWORD	- VARIABLE INDICATING THE CURRENT END MARK OF A SPEECH FILE.
WORDLENGTH	- NUMBER OF PHONEMES IN A WORD'S PHONEME REPRESENTATION.
WNUMB,WORDNO	- NUMBER CORRESPONDING TO THE WORD BEING OPERATED ON.
WORDFLAG,WORDCNT	- NUMBER CORRESPONDING TO THE NUMBER OF THE SPEECH FILE THAT WAS LAST OPERATED ON BY AN AUTOMATED PROCEDURE (RECOGALL).
NOPHVECTORS	- NO OF VECTORS TO BE USED IN DETERMINING A WORD'S PHONEME REPRESENTATION. INDICATES THE NUMBER OF VECTORS PER WORD PHONEME.
OPERATION	- OPERATION TO BE PERFORMED.
OPERATE	- ARRAY USED BY THE AUTOMATED PROCEDURES TO INDICATE THE WORDS TO BE OPERATED ON.

FUZZY VARIABLES

VARIABLE	DESCRIPTION
STHR	- DETERMINES WHETHER OR NOT A SUBSTITUTION

ERROR OCCURRED.

SUBS,SUBF	- VARIES THE EFFECT OF A SUBSTITUTION ERROR ON THE WORD SCORE.
INSE,INSF	- VARIES THE EFFECT OF AN INSERTION ERROR ON THE WORD SCORE.
DELE,DELF	- VARIES THE EFFECT OF THE STATISTICS ON THE NUMBER OF TIMES A WORD PHONEME HAS BEEN DELETED.
DELG	- WEIGHTS THE EFFECT OF THE OVERALL DELETION SCORE VERSUS PWORD.
DCNE,DCNF	- VARIES THE EFFECT OF THE TOTAL DELETION SCORE ON THE OVERALL WORD SCORE.
DCNG	- WEIGHTS THE EFFECT OF DELW VERSUS DELCNT.
SFE,SFF	- VARIES THE EFFECT OF THE SCALE FACTOR (SFACTOR) ON THE PHONEME WEIGHT(CHVALUE).
CHVE,CHVF	- VARIES THE EFFECTS OF THE WEIGHTS OF THE GUESSES PROVIDED FOR EACH VECTOR OF SPEECH.
STATE,STATF	- VARIES THE EFFECT OF THE STATISTICS COLLECTED FOR SUBSTITUTION AND INSERTION ERRORS.
STATG	- WEIGHTS THE EFFECT OF THE STATISTICS VERSUS THE PHONEME SCORE.
THR1E,THR1F, THR2E,THR2F	- USED FOR DETERMINING TRANSITIONS BETWEEN WORD PHONEMES.

NOTE: REFER TO PROCEDURE SCOREWORD TO DETERMINE THE EXACT EFFECT OF EACH OF THESE VARIABLES.

*****)

PROGRAM LEARNWORD (INPUT,OUTPUT,SPECH,WFILE,SFILE,OUT2,RESULTS,
STATSOUT,FUZZYVAR,OPTFUZZY,FZOPT);

CONST

SPELLENGTH=20;
MAXPHONES=20;
MAXPHPLUS1=21;
NOPHONEMES=71;
NOGUESSES=5;
MAXNWORDS=50;
RCRDLENGTH=40 (RCRDLENGTH MUST BE A MULTIPLE OF TWENTY);
MAXVECTORS=80;
NOTOAVG=5;
BEGINWORD=1;
NOFUZZYVARS=22;

TYPE

WORDDATA=RECORD
SPELLING:STRING[SPELLENGTH];
PHREP:ARRAY[1..MAXPHONES] OF INTEGER;

```

WSTHR,WSUBE,WSUBF,WINSE:REAL;
WINSF,WDELE,WDELF,WDELG:REAL;
WDCNE,WDCNF,WDCNG:REAL;
WSFE,WSFF,WCHVE,WCHVF:REAL;
WSTATE,WSTATF,WSTATG:REAL;
WTHR1E,WTHR1F,WTHR2E,WTHR2F:REAL;
END (*WORDDATA*);

```

```

WORDSTATS=RECORD
  NOSPOKE,NOATTEMPT,NOCORRECT:INTEGER;
  NDEL:ARRAY[1..MAXPHONES] OF INTEGER;
  XSUBY,XINFTZ:ARRAY[1..MAXPHONES,1..NOPHONEMES] OF REAL;
  TOTALSCORE:REAL;
  TOTDIFF:REAL;
  MINDIFF:REAL;
END (*WORDSTATS*);

```

```

STATSFIL=FILE OF WORDSTATS;

```

```

ISTRING=RECORD
  SPELL:STRING[SPELLENGTH];
  CONTFLAG:INTEGER;
  SPKR:STRING[10];
  ISCORE:ARRAY[1..RCRDLENGTH] OF REAL;
  IPHONE:ARRAY[1..RCRDLENGTH] OF INTEGER;
  IERROR:ARRAY[1..RCRDLENGTH] OF STRING[5];
  ICHPHONE:ARRAY[1..NOGUESSES,1..RCRDLENGTH] OF INTEGER;
  ICHVALUE:ARRAY[1..NOGUESSES,1..RCRDLENGTH] OF REAL;
  ISFACTOR:ARRAY[1..RCRDLENGTH] OF REAL
END (*ISTRING*);

```

```

WORDSTRING=FILE OF ISTRING;
WORDFILE= FILE OF WORDDATA;

```

```

FUZZYLIMITS=RECORD
  FUZNAME:ARRAY[1..NOFUZZYVARS] OF STRING[5];
  FUZVAR,MINLIMIT,MAXLIMIT:ARRAY[1..NOFUZZYVARS] OF REAL;
  RECSCORE:REAL;
END(*FUZZYLIMITS*);

```

```

FUZOPT=FILE OF FUZZYLIMITS;

```

```

VAR

```

```

  OPTION:INTEGER;
  STOPFLAG:BOOLEAN;
  CHNGEFLAG:BOOLEAN;

```

```

  WORDSCORE:REAL;

```

```

DELCNT:INTEGER;
DELM:REAL;
SCORECOUNT:INTEGER
INSRTSCORE,PWORD:REAL           ;           {WORD SCORE};
WDATA:WORDDATA;
WSTATS:WORDSTATS;
SFILE:STATSFIL;
SPECH:WORDSTRING;
WFILE:WORDFILE;
IOUT:ISTRING;
FZOPT:FUZOPT;
FLIMIT:FUZZYLIMITS;
BEGINREC:INTEGER;
IRECORD:INTEGER;
WORD:STRING[SPELLENGTH];
ENDFLAG:BOOLEAN;

```

```

RESULTS:TEXT;
STATSOUT:TEXT;
FUZZYVAR:TEXT;
OUT2:TEXT;
OPTFUZZY:TEXT;

```

```

SPEAKER:STRING[3];
FILENAME:STRING[22];
ENDWORD:INTEGER;
WORDLENGTH:INTEGER;
WNUMB,WORDNO:INTEGER;
WORDFLAG,WORDCNT:INTEGER;
NOPHVECTORS:INTEGER;
OPERATION:STRING[8];
OPERATE:ARRAY[1..MAXNOWORDS] OF STRING[3];

```

```

STHR,SUBE,SUBF:REAL;
INSE,INSF:REAL;
DELE,DELF,DELG:REAL;
DCNE,DCNF,DCNG:REAL;
SFE,SFF:REAL;
CHVE,CHVF:REAL;
STATE,STATF,STATG:REAL;
THR1E,THR1F,THR2E,THR2F:REAL;

```

```

FUNCTION POWER(ARG,PWR:REAL):REAL; FORWARD;
PROCEDURE INITFILE; FORWARD;
PROCEDURE GETFUZZYVARS; FORWARD;
PROCEDURE GETWORDVARS; FORWARD;
SEGMENT PROCEDURE INITSTAT(NOWORD:INTEGER); FORWARD;
SEGMENT PROCEDURE WINIT1; FORWARD;
SEGMENT PROCEDURE STOREWORD; FORWARD;

```

```

      { $I    SCORE.PA    }
      { $I    UPDATE.PA   }
      { $I    ISOKECOG.PA }
      { $I    CHNGEREP.PA }
      { $I    AUTODET.PA  }
      { $I    CHNGEFUZ.PA }
      { $I    AUTOMAT.PA  }

```

(*****)

PROCEDURE WINIT1 DESCRIPTION

THIS PROCEDURE ADDS A NEW SPEECH FILE TO THE EXISTING TRAINING FILES STORED IN FILE SPECH FOR A GIVEN WORD. IF THE WORD IS NOT IN THE CURRENT VOCABULARY, THIS PROCEDURE ADDS THE WORD TO THE VOCABULARY, AND INITIALIZES THE NECESSARY DATA STRUCTURES FOR THE WORD.

THE USER IS REQUIRED TO ENTER THE SPELLING OF THE WORD, AND THE NAME OF THE SPEECH FILE TO BE STORED. ALSO, IF THE WORD IS NOT IN THE CURRENT VOCABULARY, THE USER MUST ENTER AN INITIAL PHONEME REPRESENTATION FOR THE WORD (THIS REPRESENTATION DOES NOT HAVE TO BE VALID).

SINCE IT IS VITAL THAT THE WORD'S SPELLING GIVEN IN THE SPEECH FILE IS ACCURATE, A CHECK IS PERFORMED TO ASSURE THAT THIS SPELLING AGREES WITH THE SPELLING INITIALLY ENTERED BY THE USER.

(*****)

SEGMENT PROCEDURE WINIT1;

VAR

```

W,S:INTEGER;
I,J,INT:INTEGER;
TEMPWORD:STRING[SPLENGTH];
ANSWER:STRING[3];
ANS:INTEGER;
FLAG:BOOLEAN;
NAME:STRING[20];

```

BEGIN(*INIT1*)

```
ENDFLAG:=FALSE;
WRITELN;
```

```
{Get word to be learned and determine if it
is in the current vocabulary. }
```

```
WRITELN ('INPUT EXACT SPELLING OF WORD TO BE LEARNED. ');
```

```
WRITE ('WORD = ');
```

```
READLN (WORD);
```

```
FOR I:=(LENGTH(WORD)) TO (SPELLENGTH-1) DO
```

```
  BEGIN(*FOR*)
```

```
    WORD:=CONCAT(WORD,' ');
```

```
  END(*FOR*);
```

```
WRITELN ;
```

```
RESET(SFILE);
```

```
RESET (WFILE);
```

```
TEMPWORD:=' ';
```

```
WORDNO:=0;
```

```
WHILE (NOT EOF(WFILE))AND(WORD<>TEMPWORD) DO
```

```
  BEGIN (*WHILE*)
```

```
    SEEK(WFILE,WORDNO);
```

```
    GET(WFILE);
```

```
    WDATA:=WFILE^;
```

```
    GET(WFILE);
```

```
    TEMPWORD:=WDATA.SPELLING;
```

```
    WORDNO:=WORDNO+ 1
```

```
  END (*WHILE*);
```

```
{If word does not exist, and user wants to add it,
initialize the word's data structures.}
```

```
IF (WORD<>TEMPWORD) THEN
```

```
  BEGIN (*IF*)
```

```
    WRITELN;
```

```
    WRITELN ('WORD DOES;NOT EXIST. IF YOU WOULD LIKE TO ADD THIS ');
```

```
    WRITELN ('WORD TO THE VOCABULARY, ENTER "YES", ELSE ENTER "NO".');
```

```
    WRITE ('ANSWER = ');
```

```
    READLN (ANSWER);
```

```
    IF ANSWER<>'YES' THEN
```

```
      BEGIN (*IF*)
```

```
        ENDFLAG:=TRUE;
```

```
      END (*IF*)
```

```
    ELSE
```

```
      BEGIN (*ELSE*)
```

```
        {Initialize the word fuzzy variables by setting them
        equal to the overall variable values.}
```

```
        WITH WDATA DO
```

```
          BEGIN (*WITH*)
```

```
            SPELLING:=WORD;
```

```
            GETFUZZYVARS;
```

```
            WSTHR:=STHR;
```

```
            WSUBE:=SUBE;
```

```

WSUBP:=SUBP;
WINSE:=INSE;
WINSF:=INSF;
WDELE:=DELE;
WDELF:=DELF;
WDELG:=DELG;
WDCNE:=DCNE;
WDCNF:=DCNF;
WDCNG:=DCNG;
WSFE:=SFE;
WSFF:=SFF;
WCHVE:=CHVE;
WCHVF:=CHVF;
WSTATE:=STATE;
WSTATF:=STATF;
WSTATG:=STATG;
WTHRIE:=THRIE;
WTHR1F:=THR1F;
WTHR2E:=THR2E;
WTHR2F:=THR2F;
WRITELN;
WRITELN;

```

(Get the word's phoneme representation. Note that the representation does not have to be valid.)

```

WRITELN ('ENTER THE NUMERIC VALUE (01 TO 71) FOR ');
WRITELN ('EACH PHONEME IN THE WORD, UP TO A MAXIMUM OF ');
WRITE (MAXPHONES);
WRITELN (' PHONEMES. AFTER ALL WORD PHONEMES HAVE BEEN ');
WRITELN ('ENTERED, ENTER "00". ');
W:=1;
ANS:=71;
WHILE (W<=MAXPHONES)AND(ANS<>00) DO
BEGIN (*WHILE*)
WRITELN;
WRITE ('WORD PHONEME ',W:2,' = ');
READLN (ANS);
IF (00<=ANS)AND(ANS<=71) THEN
BEGIN (*IF*)
PHREPL[W]:=ANS;
W:=W+1
END (*IF*)
ELSE
BEGIN (*ELSE*)
WRITELN;
WRITELN ('PHONEME NOT VALID, REENTER');
END (*ELSE*)
END (*WHILE*);

```

```

IF PHREPL[1] = 00 THEN

```

```

        ENDFLAG:=TRUE
ELSE
BEGIN (*ELSE*)
    IF W(MAXPHONES THEN
    BEGIN (*IF*)
        FOR J:= W TO MAXPHONES DO
            PHREPLJJ:=00
        END (*FOR*);
    END (*IF*);
    IF WORDNO=0 THEN
    BEGIN(*IF*)
        APPEND(WFILE);
    END (*IF*)
    ELSE
    BEGIN(*ELSE*)
        SEEK(WFILE,(WORDNO-1));
        GET(WFILE);
    END(*ELSE*);
    WFILE^:=WDATA;
    PUT (WFILE);

    {Initialize the word statistics.}

    WITH WSTATS DO
    BEGIN (*WITH*)
        NOSPOKE:=0;
        NOCORRECT:=0;
        NOATTEMPT:=0;
        TOTDIFF:=0.0;
        MINDIFF:=1.0;
        FOR I:= 1 TO MAXPHONES DO
        BEGIN (*FOR*)
            NDEL[I]:=0;
            FOR J:= 1 TO NOPHONEMES DO
            BEGIN (*FOR*)
                XINAFTZ[I,J]:=0.0;
                XSUBY[I,J]:=0.0
            END (*FOR*)
        END (*FOR*);
        TOTALSCORE:=0.0;
        IF WORDNO=0 THEN
        BEGIN(*IF*)
            APPEND(SFILE);
        END (*IF*)
        ELSE
        BEGIN(*ELSE*)
            SEEK(SFILE,(WORDNO-1));
            GET(SFILE);
        END(*ELSE*);
        SFILE^:=WSTATS;
        PUT (SFILE)
    END (*WITH*)

END (*ELSE*)

```

```

        END (*WITH*)
    END (*ELSE*)
END (*IF*);

IF ENDFLAG = FALSE THEN
BEGIN (*IF*)

    (Get speech file to be used for training, and assure
    that the speaker and word spelling are valid.)

```

```

    RESET(SPECH);
    WRITELN;
    WRITELN('INPUT NAME OF SPEECHFILE');
    WRITE('FILENAME = ');
    READLN(NAME);
    WRITELN;
    FLAG:=TRUE;
    INT:=0;
    J:=1;
    WHILE J(<)LENGTH(NAME) DO
    BEGIN (*WHILE*)
        J:=J+1;
        IF NAME[J]=':' THEN
        BEGIN (*IF*)
            INT:=J;
        END(*IF*);
    END(*WHILE*);
    FOR I:=1 TO 3 DO
    BEGIN(*FOR*)
        IF SPEAKER[I](<) NAME[I+INT] THEN
            FLAG:=FALSE;
    END (*FOR*);
    IF ((FLAG=FALSE)AND(SPEAKER(<)'ALL')) THEN
    BEGIN(*IF*)
        WRITELN;
        WRITELN;
        WRITELN('SPEAKER'S INITIALS DO NOT AGREE WITH THE ');
        WRITELN('FIRST 3 LETTERS OF THE SPEECHFILE. DO YOU ');
        WRITELN('WISH TO CONTINUE ANYWAY! ENTER "YES" TO');
        WRITELN('CONTINUE, ELSE ENTER "NO".');
        WRITE('ANSWER = ');
        READLN(ANSWER);
        WRITELN;
        IF ANSWER(<) 'YES' THEN
        BEGIN(*IF*)
            ENDFLAG:=TRUE;
        END(*IF*);
    END (*IF*);
    IF ENDFLAG=FALSE THEN
    BEGIN(*IF*)
        FILETITLE(OUT2,NAME);
        RESET (OUT2);
        WITH IOUT DO

```

```

BEGIN (*WITH*)
  READLN(OUT2, SPELL);
  FOR I:= (LENGTH(SPELL)) TO (SPELLENGTH-1) DO
  BEGIN (*FOR*)
    SPELL:=CONCAT(SPELL, ' ');
  END (*FOR*);
  WORD:=WDATA.SPELLING;
  IF SPELL<>WORD THEN
  BEGIN (*IF*)
    FLAG:=FALSE;
    REPEAT
      WRITELN;
      WRITELN ('THE SPELLING OF THE WORD GIVEN ABOVE DOES');
      WRITELN ('NOT AGREE WITH THE SPELLING GIVEN IN THE');
      WRITE ('FILE "', NAME, ". THE SPELLING IN ', NAME, ' IS');
      WRITELN (' ', SPELL, ' ');
      WRITELN ('DO YOU WANT TO CHANGE THE "', NAME, " SPELLING?');
      WRITELN ('ENTER "YES" TO CHANGE, ELSE ENTER "NO"');
      WRITE ('ANSWER = ');
      READLN(ANSWER);
      IF ANSWER<>'YES' THEN
      BEGIN (*IF*)
        ENDFLAG:=TRUE;
      END (*IF*)
      ELSE
      BEGIN (*ELSE*)
        WRITELN;
        WRITELN ('ENTER THE CORRECT SPELLING OF WORD. ');
        WRITE ('WORD = ');
        READLN (SPELL);
        FOR I:= (LENGTH(SPELL)) TO (SPELLENGTH-1) DO
        BEGIN (*FOR*)
          SPELL:=CONCAT(SPELL, ' ');
        END (*FOR*);
      END (*ELSE*);
      IF (ENDFLAG=TRUE) OR (SPELL=WDATA.SPELLING) THEN
      BEGIN (*IF*)
        FLAG:=TRUE;
      END (*IF*);
    UNTIL FLAG;
  END (*IF*);
  END(*WITH*);
END(*IF*);

END (*IF*);

END (*PROCEDURE INIT1*);

```

(*****)

PROCEDURE STOREWORD DESCRIPTION

THIS PROCEDURE DIVIDES A SPEECH FILE, REFERRED TO AS OUT2, INTO RECORDS OF THE FORMAT REQUIRED BY FILE SPECH, AND STORES THESE RECORDS IN THE FILE SPECIFIED AS SPECH. NOTE THAT THIS PROCEDURE ASSUMES THAT THE WORD SPELLING, STORED IN THE SPEECH FILE, HAS ALREADY BEEN READ.

(*****)

SEGMENT PROCEDURE STOREWORD;

VAR

TEMPIOUT:ISTRING;
INT,I,J:INTEGER;

BEGIN (*PROCEDURE STOREWORD*)

WITH IOUT DO
BEGIN(*WITH*)
 READLN(OUT2,SPKR);
 FOR I:=1 TO 8 DO
 BEGIN (*FOR*)
 READLN(OUT2);
 END (*FOR*);
 IRECORD:=0;

{Insert first record of speech file at end
of file SPECH.}

RESET(SPECH);
WHILE NOT EOF(SPECH) DO
BEGIN(*WHILE*)
 SEEK(SPECH,IRECORD);
 GET (SPECH);
 TEMPIOUT:=SPECH^;
 IRECORD:=IRECORD+1;
 GET(SPECH);
END (*WHILE*);
BEGINREC:=IRECORD;

{Divide speech file into records, and
initialize the data structures.}

J:=0;
WHILE NOT EOF(OUT2) DO

```

BEGIN (*WHILE*)
  J:=J+1;
  IF J=(RCRDLENGTH+1) THEN
    BEGIN(*IF*)
      CONTFLAG:=1;
      IF IRECORD=0 THEN
        BEGIN(*IF*)
          APPEND(SPECH);
        END (*IF*)
      ELSE
        BEGIN(*ELSE*)
          SEEK(SPECH,(IRECORD-1));
          GET(SPECH);
        END(*ELSE*);
        IRECORD:=IRECORD+1;
        SPECH^:=IOUT;
        PUT(SPECH);
        J:=1;
      END (*IF*);
      READ(OUT2,INT);
      FOR I:=1 TO NOGUESSES DO
        BEGIN(*FOR*)
          READ (OUT2,ICHPHONE[I,J]);
          READ (OUT2,ICHVALUE[I,J]);
        END (*FOR*);
        READLN (OUT2,ISFACTOR[J]);
        ISCORE[J]:=0.0;
        IPHONE[J]:=0;
        IERROR[J]:='XXXXX';
      END (*WHILE*);

      CONTFLAG:=0;
      FOR INT:=(J+1) TO RCRDLENGTH DO
        BEGIN (*FOR*)
          FOR I:=1 TO NOGUESSES DO
            BEGIN (*FOR*)
              ICHPHONE[I,INT]:=0;
              ICHVALUE[I,INT]:=1.0;
            END (*FOR*);
            ISFACTOR[INT]:=0.0;
            ISCORE[INT]:=0.0;
            IPHONE[INT]:=0;
            IERROR[INT]:='XXXXX';
          END (*FOR*);
          IF IRECORD=0 THEN
            BEGIN(*IF*)
              APPEND(SPECH);
            END (*IF*)
          ELSE
            BEGIN(*ELSE*)
              SEEK(SPECH,(IRECORD-1));
              GET(SPECH);
            END(*ELSE*);
            IRECORD:=IRECORD+1;

```

```

        SPECH^:=1OUT;
        PUT(SPECH);

    END(*WITH*);
    CLOSE(OUT2);

END(*PROCEDURE STOREWORD*);

```

```

{*****}

```

PROCEDURE DETPHREP DESCRIPTION

THIS PROCEDURE DETERMINES THE PHONEME REPRESENTATION FOR A GIVEN WORD BASED ON THE THE WORD'S TRAINING SET STORED IN FILE SPECH. BASICALLY, THIS PROCEDURE MAKES AN INITIAL EDUCATED GUESS OF THE PHONEME REPRESENTATION, THEN ITERATIVELY ALTERS THIS REPRESENTATION BASED ON THE STATISTICS OBTAINED FROM ATTEMPTING TO RECOGNIZE THE WORD USING THE TRAINING SET SPEECH FILES.

THE USER MUST ENTER THE WORD TO BE OPERATED ON, THE NUMBER OF VECTORS PER WORD PHONEME, AND THE MAXIMUM NUMBER OF ITERATIONS (NOITERATIONS) TO BE PERFORMED. REFER TO THE DESCRIPTION GIVEN FOR PROCEDURE AUTODET FOR A MORE DETAILED DISCUSSION OF THE ALGORITHM EMPLOYED.

```

*****}

```

```

SEGMENT PROCEDURE DETPHREP;

```

```

VAR
    NOITERATIONS,ICOUNT,I,NOWORD:INTEGER;
    TEMPWORD,DETWORD:STRING[SPLENGTH];

```

```

BEGIN (*DETPHREP*)

```

```

    INITFILE;
    GETFUZZYVARS;
    RESET(WFILE);
    RESET(SFILE);
    WORDNO:=0;

```

```

NOWORD:=0;
SEEK(WFILE,NOWORD);
GET(WFILE);
Writeln;
Writeln('THE FOLLOWING IS THE CURRENT VOCABULARY');
Writeln;
WHILE NOT EOF(WFILE) DO
BEGIN (*WHILE*)
    WDATA:=WFILE^;
    Writeln(WDATA.SPELLING);
    GET(WFILE);
END (*WHILE*);
Writeln;
Writeln;
Writeln('INPUT WORD TO DETERMINE PHONEME REPRESENTATION');
Writeln;
WRITE ('WORD = ');
READLN (DETWORD);
Writeln;
FOR I:=(LENGTH(DETWORD)) TO (SPELLENGTH-1) DO
BEGIN(*FOR*)
    DETWORD:=CONCAT(DETWORD,' ');
END(*FOR*);
RESET (WFILE);
TEMPWORD:=' ';
NOWORD:=0;
WHILE (NOT EOF(WFILE))AND(DETWORD<>TEMPWORD) DO
BEGIN (*WHILE*)
    SEEK(WFILE,NOWORD);
    GET(WFILE);
    WDATA:=WFILE^;
    GET(WFILE);
    TEMPWORD:=WDATA.SPELLING;
    NOWORD:=NOWORD+ 1
END (*WHILE*);
IF (DETWORD<>TEMPWORD) THEN
BEGIN (*IF*)
    Writeln;
    Writeln('WORD DOES NOT EXIST. ');
    ENDFLAG:=TRUE;
END (*IF*)
ELSE
BEGIN(*ELSE*)
    ENDFLAG:=FALSE;
    NOWORD:=NOWORD-1;
    Writeln;
    Writeln;
    Writeln('ENTER THE NUMBER OF VECTORS PER WORD PHONEME. ');
    WRITE(' * VECTORS = ');
    READ(NOPHVECTORS);
    Writeln;
    Writeln('ENTER THE NUMBER OF ITERATIONS TO BE PERFORMED. ');
    WRITE('NO. ITERATIONS = ');
    READLN(NOITERATIONS);

```

```

WRITELN;
FILETITLE(STATSOUT,'PHREPDAT');
REWRITE(STATSOUT);

OPERATION:='AUTODET';
AUTODET(NOWORD,NOPHVECTORS,CHNGEFLAG);
INITSTAT(NOWORD);
ICOUNT:=0;
REPEAT
    ICOUNT:=ICOUNT+1;
    CHNGEFLAG:=FALSE;
    OPERATION:='CORRECTR';
    AUTODET(NOWORD,NOPHVECTORS,CHNGEFLAG);
    INITSTAT(NOWORD);
    OPERATION:='ADJUSTDE';
    AUTODET(NOWORD,NOPHVECTORS,CHNGEFLAG);
    INITSTAT(NOWORD);
UNTIL (NOITERATIONS=ICOUNT)OR(CHNGEFLAG=FALSE);
CLOSE(STATSOUT);
END(*ELSE*);

```

```

END(*PROCEDURE DETPHREP*);

```

```

{*****}

```

PROCEDURE INITSTAT DESCRIPTION

```

-----

THE PURPOSE OF THIS PROCEDURE IS TO CALCULATE A WORD'S STATISTICS
USING THE WORD'S TRAINING SET PROVIDED IN FILE SPECH. THIS IS DONE BY
OBTAINING THE ERRORS THAT OCCUR IN ATTEMPTING TO RECOGNIZE EACH
OF THE TRAINING SET FILES (BY EXECUTING PROCEDURE SCOREWORD), AND
USING THIS INFORMATION TO UPDATE THE STATISTICS (BY EXECUTING PROCEDURE
UPDATE).

```

```

*****}

```

```

SEGMENT PROCEDURE INITSTAT;

VAR
    J,INT:INTEGER;

BEGIN(*INITSTAT*)

    WRITELN;
    WRITELN('INITSTAT');

    {Initialize word statistics.}

    WITH WSTATS DO
    BEGIN (*WITH*)
        NOSPOKE:=0;
        NOCORRECT:=0;
        NOATTEMPT:=0;
        TOTDIFF:=0.0;
        MINDIFF:=1.0;
        FOR INT:= 1 TO MAXPHONES DO
        BEGIN (*FOR*)
            NDELI[INT]:=0;
            FOR J:= 1 TO NOPHONEMES DO
            BEGIN (*FOR*)
                XINFTZ[INT,J]:=0.0;
                XSUBY[INT,J]:=0.0
            END (*FOR*)
        END (*FOR*);
        TOTALSCORE:=0.0;
        SEEK(SFILE,NOWORD);
        SFILE^:=WSTATS;
        PUT (SFILE)
    END (*WITH*);

    SEEK(WFILE,NOWORD);
    GET(WFILE);
    WDATA:=WFILE^;
    WRITELN('WORD = ',WDATA.SPELLING);
    WORD:=WDATA.SPELLING;
    BEGINREC:=0;

    {Update statistics using the word's training
    files stored in file SPECH.}

    RESET(SPECH);
    WHILE NOT EOF(SPECH) DO
    BEGIN(*WHILE*)
        SEEK(SPECH,BEGINREC);
        GET(SPECH);
        IOUT:=SPECH^;
        IF IOUT.SPELL=WORD THEN
        BEGIN(*IF*)
            SCOREWORD;
            UPDATE;
        END(*IF*);
    END(*WHILE*);

```

```

        SEEK(SPECH,BEGINREC);
        REPEAT
            GET(SPECH);
            IOUT:=SPECH^;
            BEGINREC:=BEGINREC+1;
        UNTIL (IOUT.CONTFLAG=0);
        GET(SPECH);
    END(*IF*)
    ELSE
    BEGIN(*ELSE*)
        BEGINREC:=BEGINREC+1;
        GET(SPECH);
    END(*ELSE*);

    END(*WHILE*);

END(*PROCEDURE INITSTAT*);

```

{*****}

FUNCTION POWER DESCRIPTION

THIS FUNCTION IS PASSED TWO VARIABLES, ARG AND PWR, WHICH IT USES TO CALCULATE THE VALUE OF POWER, WHICH IS EQUAL TO "ARG" RAISED TO THE "PWR" POWER.

*****}

```

FUNCTION POWER;

BEGIN(*POWER*)

    IF ((ARG)=0)AND(ARG<0.0001)) THEN
        POWER:=0
    ELSE
        POWER:=EXP(PWR*(LN(ARG)));

END (*POWER*);

```

{*****}

PROCEDURE INITFILE DESCRIPTION

THIS PROCEDURE ACCEPTS THE THREE INITIALS PRECEEDING THE FILES TO BE OPERATED UPON, AND CONCATINATES THEM WITH THE APPROPRIATE FILE NAMES. THE FOLLOWING FILES ARE AFFECTED:

WFILE
SFILE
SPECH
FZOPT

{*****}

PROCEDURE INITFILE;

BEGIN (*INITFILE*)

```
WRITELN;  
WRITELN;  
WRITELN('ENTER "THREE" INITIALS FOR THE CURRENT *PERMANENT* ');  
WRITELN('SPEAKER; OR ENTER "ALL" FOR *NON-PERMANENT* SPEAKERS.');
```

WRITE('SPEAKER = ');
READLN(SPEAKER);
WRITELN;
FILENAME:=CONCAT(SPEAKER,'WFILE');
FILETITLE(WFILE,FILENAME);
FILENAME:=CONCAT(SPEAKER,'SFILE');
FILETITLE(SFILE,FILENAME);
FILENAME:=CONCAT(SPEAKER,'SPECH');
FILETITLE(SPECH,FILENAME);
FILENAME:=CONCAT(SPEAKER,'FZOPT');
FILETITLE(FZOPT,FILENAME);

END(*PROCEDURE INITFILE*);

{*****}

PROCEDURE GETFUZZYVARS DESCRIPTION

THIS PROCEDURE READS THE VALUES OF THE OVERALL FUZZY VARIABLES
STORED IN THE FILE FUZZYVAR.

*****)

PROCEDURE GETFUZZYVARS;

BEGIN(*GETFUZZYVARS*);

RESET(FUZZYVAR);
READLN(FUZZYVAR,STHR,SUBE,SUBF);
READLN(FUZZYVAR,INSE,INSF);
READLN(FUZZYVAR,DELE,DELF,DELG);
READLN(FUZZYVAR,DCNE,DCNF,DCNG);
READLN(FUZZYVAR,SFE,SFF);
READLN(FUZZYVAR,CHVE,CHVF);
READLN(FUZZYVAR,STATE,STATF,STATG);
READLN(FUZZYVAR,THR1E,THR1F);
READLN(FUZZYVAR,THR2E,THR2F);

CLOSE(FUZZYVAR);

END(*PROCEDURE GETFUZZYVARS*);

{*****)

PROCEDURE GETWORDVARS DESCRIPTION

THIS PROCEDURE INITIALIZES THE VALUES OF THE GLOBAL FUZZY
VARIABLES TO CORRESPOND TO A WORD'S FUZZY VARIABLES. RECORD
WDATA MUST CONTAIN THE APPROPRIATE WORD'S DATA BEFORE THIS
PROCEDURE IS CALLED.

*****)

PROCEDURE GETWORDVARS;

BEGIN(*GETWORDVARS*)

```

WITH WDATA DO
BEGIN(*WITH*)
    STHR:=WSTHR;
    SUBE:=WSUBE;
    SUBF:=WSUBF;
    INSE:=WINSE;
    INSF:=WINSF;
    DELE:=WDELE;
    DELF:=WDELF;
    DELG:=WDELG;
    DCNE:=WDCNE;
    DCNF:=WDCNF;
    DCNG:=WDCNG;
    SFE:=WSFE;
    SFF:=WSFF;
    CHVE:=WCHVE;
    CHVF:=WCHVF;
    STATE:=WSTATE;
    STATF:=WSTATF;
    STATG:=WSTATG;
    THR1E:=WTHR1E;
    THR1F:=WTHR1F;
    THR2E:=WTHR2E;
    THR2F:=WTHR2F;

    END(*WITH*);

END(*PROCEDURE GETWORDVARS*);

```

```

BEGIN (*PROGRAM LEARNWORDS*)

```

```

    STOPFLAG:=FALSE;
    REPEAT
        WRITELN;
        WRITELN;
        WRITELN('CHOOSE OPTION FROM FOLLOWING LIST');
        WRITELN;
        WRITELN('1 - COLLECT STATISTICS ON A SPEECH FILE. ');
        WRITELN('2 - ISOLATED WORD RECOGNITION. ');
        WRITELN('3 - DETERMINE PHONEME REPRESENTATION FOR A GIVEN WORD. ');
        WRITELN('4 - CHANGE PHONEME REPRESENTATION OF A GIVEN WORD. ');
        WRITELN('5 - CHANGE THE FUZZY VARIABLES. ');
        WRITELN('6 - RUN AN AUTOMATIC PROGRAM. ');
        WRITELN('9 - STOP. ');
        WRITELN;
        WRITE('OPTION = ');
        READLN(OPTION);
        WRITELN;
    
```

```

IF OPTION = 1 THEN
BEGIN(*IF*)
  INITFILE;
  WINIT1;
  IF ENDFLAG=FALSE THEN
  BEGIN (*IF*)
    STOREWORD;
    GETFUZZYVARS;
    SCOREWORD;
    UPDATE;
  END(*IF*);
END(*IF*)
ELSE IF OPTION = 2 THEN
BEGIN(*ELSE*)
  ISORECOG;
END(*ELSE*)
ELSE IF OPTION = 3 THEN
BEGIN(*ELSE*)
  DETPHREP;
END(*ELSE*)
ELSE IF OPTION = 4 THEN
BEGIN(*ELSE*)
  CHNGEREP;
END(*ELSE*)
ELSE IF OPTION = 5 THEN
BEGIN(*ELSE*)
  CHNGEFUZ;
END(*ELSE*)
ELSE IF OPTION = 6 THEN
BEGIN(*ELSE*)
  AUTOMAT;
END(*ELSE*)
ELSE IF OPTION = 9 THEN
BEGIN(*ELSE*)
  STOPFLAG:=TRUE;
END (*ELSE*)
ELSE
BEGIN(*ELSE*)
  WRITELN;
  WRITELN('OPTION DOES NOT EXIST');
  WRITELN;
END(*ELSE*);
CLOSE(WFILE);
CLOSE(SFILE);
CLOSE(SPECH);
UNTIL STOPFLAG;

END (*PROGRAM LEARN*).

```

(*****

PROCEDURE SCOREWORD DESCRIPTION

THIS PROCEDURE USES THE WORD INFORMATION STORED IN FILES WFILE AND SFILE, TO PRODUCE A SCORE INDICATING HOW WELL THE SPEECH FILE MATCHES THE GIVEN WORD. THE SPEECH FILE BEING ANALYZED IS STORED IN EITHER FILE RECOG OR A TEMPORARY FILE(TEMPFILE). ALTHOUGH THE SPEECH FILE IS REFERRED TO AS "SPECH", THE PROPER FILE NAME IS ASSIGNED, USING THE FILETITLE COMMAND, PRIOR TO CALLING THIS PROCEDURE.

THE FOLLOWING VARIABLES MUST BE INITIALIZED PRIOR TO CALLING THIS PROCEDURE: WORD, BEGINREC, BEGINWORD, AND NOTOAVG.

PROCEDURE VARIABLES

VARIABLE	DESCRIPTION
-----	-----
W	- NUMBER INDICATING THE CURRENT WORD PHONEME.
S	- CURRENT VECTOR OF SPEECH BEING EXAMINED.
VECTOR	- VECTOR OF SPEECH BEING EXAMINED (NOT NECESSARILY THE CURRENT VECTOR.
SCORE,PHONE, VERROR,CHPHONE, CHVALUE,SFACTOR	- IDENTICAL TO THE VARIABLES DESCRIBED IN FILE SPECH'S DESCRIPTION (THE VARIABLE NAMES ARE PRECEDED BY AN "I" IN FILE SPECH) EXCEPT THESE ARRAYS CONTAIN TWICE THE NUMBER OF SPEECH VECTORS. THESE ARRAYS ARE NECESSARY TO REDUCE THE COMPUTATION TIME REQUIRED IN EXAMINING VECTORS OCCURRING NEAR THE END OF A RECORD.
OFFSET	- USED TO KEEP TRACK OF THE VECTOR NUMBER IN THE ABOVE ARRAY'S. THIS IS NECESSARY SINCE ONLY A FINITE NUMBER OF RECORDS (OF THE TYPE IOUT) CAN FIT IN MEMORY AT ONE TIME.
PR,ERKOR	- VARIABLES RETURNED FROM PROCEDURE FINDMAX INDICATING A WORD PHONEME'S SCORE, AND MAXIMUM LIKELY ERROR TO HAVE OCCURRED, RESPECTIVELY.
PHSCR,TERROR	- ARRAYS USED TO DETERMINE WHICH WORD PHONEME OCCURRED FOR A GIVEN SPEECH VECTOR.

*****}

SEGMENT PROCEDURE SCOREWORD;

{NEEDS: WORD,BEGINREC,BEGINWORD,NOTOAVG,VAR-PWORD,SCORECOUNT}

TYPE

STR5=STRING[5];

VAR

W:INTEGER {WORD PHONEME POINTER};
S:INTEGER {STRING PHONEME POINTER};
VECTOR:INTEGER;
SCORE:ARRAY[1..MAXVECTORS] OF REAL;
PHONE:ARRAY[1..MAXVECTORS] OF INTEGER;
VEERROR:ARRAY[1..MAXVECTORS] OF STRING[5];
CHPHONE:ARRAY[1..NOGUESSES,1..MAXVECTORS] OF INTEGER;
CHVALUE:ARRAY[1..NOGUESSES,1..MAXVECTORS] OF REAL;
SFACOR:ARRAY[1..MAXVECTORS] OF REAL;
I,J:INTEGER;
FLAG:BOOLEAN;
N:INTEGER;
PR:REAL;
ERROR:STRING[5];
PHSCR:ARRAY[1..MAXPHONES] OF REAL;
TERROR:ARRAY[1..MAXPHONES] OF STRING[5];
PW,PW1,PW2:REAL;
OFFSET:INTEGER;
TEMPWORD:STRING[SPELLENGTH];
NOAVG:INTEGER;

PROCEDURE WGE2SGE3; FORWARD;
PROCEDURE WEQ1SLE2; FORWARD;
PROCEDURE WEQ1SGE3; FORWARD;
PROCEDURE WGE2SLE2; FORWARD;
PROCEDURE FINDMAX(N:INTEGER; VECTOR:INTEGER;
VAR PR:REAL; VAR ERROR:STR5); FORWARD;
PROCEDURE PWISMAX; FORWARD;
PROCEDURE PW1ISMAX; FORWARD;
PROCEDURE PW2ISMAX; FORWARD;
PROCEDURE WDELETED(N:INTEGER); FORWARD;
PROCEDURE GETRECORD; FORWARD;

{*****}

PROCEDURE GET2RECORDS DESCRIPTION

THIS PROCEDURE GETS THE FIRST TWO RECORDS OF THE SPEECH FILE
(IF THE FILE IS MORE THAN ONE RECORD LONG), AND STORES THIS
INFORMATION IN THE APPROPRIATE ARRAYS.

*****}

PROCEDURE GET2RECORDS;

VAR

I,J:INTEGER;

BEGIN(*GET2RECORDS*)

{Initialize record arrays to zero.}

FOR J:=1 TO MAXVECTORS DO

BEGIN (*FOR*)

FOR I:= 1 TO NOGUESSES DO

BEGIN (*FOR*)

CHPHONE[I,J]:=0;

CHVALUE[I,J]:=0.0;

END (*FOR*);

SFACTOR[J]:=0.0;

VERROR[J]:='XXXXX';

END (*FOR*);

RESET(SPECH);

SEEK (SPECH,I RECORD);

GET (SPECH);

IOUT:=SPECH^;

{Set array values to corresponding values given
in the speech file being examined.}

WITH IOUT DO

BEGIN (*WITH*)

FOR J:=1 TO RCRDLENGTH DO

BEGIN (*FOR*)

FOR I:=1 TO NOGUESSES DO

BEGIN (*FOR*)

CHPHONE[I,J]:=ICHPHONE[I,J];

CHVALUE[I,J]:=ICHVALUE[I,J];

END (*FOR*);

```

SFACOR[J]:=ISFACOR[J];
SCORE[J]:=ISCORE[J];
PHONE[J]:=IPHONE[J];
VERROR[J]:=IVEROR[J];
IF CHPHONE[1,J] (> 0 THEN
  BEGIN (*IF*)
    ENDWORD:=J;
  END (*IF*);
END (*FOR*);
END (*WITH*);

```

{If speech file is longer than one record, initialize the second half of the array.}

```

IF IOUT.CONTFLAG = 1 THEN
  BEGIN (*IF*)
    GET (SPECH);
    IOUT:=SPECH^;
    WITH IOUT DO
      BEGIN (*WITH*)
        FOR J:=1 TO KCRDLENGTH DO
          BEGIN (*FOR*)
            FOR I:= 1 TO NOGUESSES DO
              BEGIN (*FOR*)
                CHPHONE[I,J+RCRDLENGTH]:=ICHPHONE[I,J];
                CHVALUE[I,J+RCRDLENGTH]:=ICHVALUE[I,J];
              END (*FOR*);
              SFACOR[J+RCRDLENGTH]:=ISFACOR[J];
              SCORE[J+RCRDLENGTH]:=ISCORE[J];
              PHONE[J+RCRDLENGTH]:=IPHONE[J];
              VERROR[J+RCRDLENGTH]:=IVEROR[J];
              IF ICHPHONE[1,J] (> 0 THEN
                BEGIN (*IF*)
                  ENDWORD:=J+RCRDLENGTH;
                END (*IF*);
              END (*FOR*);
            END (*WITH*);
          END (*IF*);
        END (*PROCEDURE GET2RECORDS*);

```

PROCEDURE WGE2SGE3 DESCRIPTION

THIS PROCEDURE DEKIVES ITS NAME FROM THE FACT THAT IS ONLY CALLED WHEN WORDLENGTH - W >= 2 AND ENDWORD-S >=3. IN OTHER WORDS, IT IS ONLY CALLED WHEN MORE THAN 2 PHONEMES REMAIN IN THE WORD PHONEME REPRESENTATION, AND WHEN THERE ARE MORE THAN 3 VECTORS REMAINING IN THE SPEECH FILE.

THIS PROCEDURE DETERMINES WHICH WORD PHONEME MOST LIKELY

OCCURRED FOR THE CURRENT VECTOR OF SPEECH (S). THIS IS BASICALLY ACCOMPLISHED BY EXAMINING THE PLAUSIBILITY OF THE CURRENT WORD PHONEME (W) OCCURRING, AS COMPARED WITH THE PLAUSIBILITIES OF THE NEXT TWO WORD PHONEMES (W+1 AND W+2) OCCURRING FOR THE CURRENT VECTOR OF SPEECH. IF THE CURRENT WORD PHONEME PLAUSIBILITY IS LESS THAN EITHER OF THE NEXT TWO PHONEMES, THEN THE NEXT "NOTOAVG" VECTORS (3 VECTORS IF NOTOAVG) THE NUMBER OF VECTORS LEFT) WILL BE EXAMINED TO DETERMINE IF THE CURRENT VECTOR IS AN ACTUAL TRANSITION BETWEEN WORD PHONEMES, OR IS AN INSERTION ERROR.

*****}

PROCEDURE WGE2SGE3;

VAR

VECTOR,N:INTEGER;

BEGIN (*WGE2SGE3*)

FOR N:=W TO (W+2) DO

BEGIN (*FOR*)

VECTOR:=S;

FINDMAX (N,VECTOR,PR,ERROR);

PHSCR[N]:=PR;

TERROR[N]:=ERROR;

END (*FOR*);

IF (PHSCR[W])PHSCR[W+1])AND(PHSCR[W])=PHSCR[W+2]) THEN

BEGIN (*IF*)

PWISMAX;

END (*IF*);

ELSE IF (PHSCR[W]<PHSCR[W+1])AND(PHSCR[W+1])=PHSCR[W+2]) THEN

BEGIN (*ELSE*)

PW:=PHSCR[W];

PW1:=PHSCR[W+1];

IF (ENDWORD-S-NOTOAVG) >= 0 THEN

NOAVG:=NOTOAVG

ELSE

NOAVG:=3;

FOR VECTOR:=(S+1) TO (S+NOAVG) DO

BEGIN (*FOR*)

N:=W;

FINDMAX(N,VECTOR,PR,ERROR);

PW:=PW+PR;

N:=W+1;

FINDMAX(N,VECTOR,PR,ERROR);

PW1:=PW1+PR;

END (*FOR*);

PW:=PW/(NOAVG+1);

PW1:=PW1/(NOAVG+1);

IF ((THR1E*POWER(PW,THR1F)))=PW1) THEN

```

        BEGIN (*IF*)
            PWISMAX;
        END (*IF*)
    ELSE
        BEGIN (*ELSE*)
            PW1ISMAX;
        END (*ELSE*);
    END (*ELSE*)
ELSE IF (PHSCR[W+2]>PHSCR[W])AND(PHSCR[W+2]>PHSCR[W+1]) THEN
    BEGIN (*ELSE*)
        PW:=PHSCR[W];
        PW1:=PHSCR[W+1];
        PW2:=PHSCR[W+2];
        IF (ENDWORD-S-NOTAVG) >= 0 THEN
            NOAVG:=NOTAVG
        ELSE
            NOAVG:=3;
        FOR VECTOR:=(S+1) TO (S+NOAVG) DO
            BEGIN (*FOR*)
                N:=W;
                FINDMAX(N,VECTOR,PR,ERROR);
                PW:=PW+PR;
                N:=W+1;
                FINDMAX (N,VECTOR,PR,ERROR);
                PW1:=PW1+PR;
                N:=W+2;
                FINDMAX(N,VECTOR, PR,ERROR);
                PW2:=PW2+PR;
            END (*FOR*);
            PW:=PW/(NOAVG+1);
            PW1:=PW1/(NOAVG+1);
            PW2:=PW2/(NOAVG+1);
            IF ((THR1E*POWER(PW,THR1F)))=PW1)AND
                ((THR2E*POWER(PW,THR2F)))=PW2) THEN
                BEGIN (*IF*)
                    PWISMAX;
                END (*IF*)
            ELSE IF (PW1)<(THR1E*POWER(PW,THR1F))) AND
                ((THR2E*POWER(PW1,THR2F)))=PW2) THEN
                BEGIN (*ELSE*)
                    PW1ISMAX;
                END (*ELSE*)
            ELSE
                BEGIN (*ELSE*)
                    PW2ISMAX;
                END (*ELSE*);
            END (*ELSE*);
        END (*ELSE*);
    END (*ELSE*);

```

```

END (*PROCEDURE WGE2SGE3*);

```

{*****}

PROCEDURE WEQ1SLE2 DESCRIPTION

THIS PROCEDURE IS CALLED WHEN THERE ARE ONLY TWO WORD PHONEMES REMAINING AND WHEN THERE ARE LESS THAN 3 VECTORS OF SPEECH REMAINING. UNLIKE THE PREVIOUS PROCEDURE, THE DECISION OF WHETHER OR NOT A TRANSITION OCCURRED IS BASED SOLEY ON THE PLAUSIBILITY OF THE REMAINING PHONEMES OCCURRING FOR THE CURRENT VECTOR. IF A TRANSITION OCCURS, THEN THE PLAUSIBILITIES OF THE LAST PHONEME OCCURRING FOR THE REMAINING VECTORS OF SPEECH ARE COMPUTED.

*****}

PROCEDURE WEQ1SLE2;

VAR

N:INTEGER;

BEGIN (*WEQ1SLE2*)

FOR N:=W TO (W+1) DO

BEGIN (*FOR*)

VECTOR:=S;

FINDMAX (N,VECTOR,PR,ERROR);

TERROR[N]:=ERROR;

PHSCR[N]:=PR;

END (*FOR*);

IF PHSCR[W]>PHSCR[W+1] THEN

BEGIN (*IF*)

PWISMAX;

IF (ENDWORD-S)<0 THEN

BEGIN (*IF*)

N:=W+1;

WDELETED(N);

END (*IF*)

END (*IF*)

ELSE IF PHSCR[W+1]=PHSCR[W] THEN

BEGIN (*ELSE*)

PW11SMAX;

WHILE (ENDWORD-S)>=0 DO

BEGIN (*WHILE*)

VECTOR:=S;

FINDMAX(W,VECTOR,PR,ERROR);

PHSCR[W]:=PR;

```

        TERROR[W]:=ERROR;
        PWISMAX;
    END (*WHILE*);
END (*ELSE*);

```

```

END (*WEQ1SLE2*);

```

```

{*****}

```

PROCEDURE WEQ1SGE3 DESCRIPTION -----

THIS PROCEDURE IS ONLY CALLED WHEN THERE ARE TWO WORD PHONEMES LEFT AND WHEN THERE ARE MORE THAN 2 VECTORS OF SPEECH REMAINING. THIS PROCEDURE IS SIMILAR TO PROCEDURE WGE2SGE3, EXCEPT THAT WHEN A TRANSITION TO THE LAST WORD PHONEME OCCURS, THE PLAUSIBILITIES FOR THE LAST WORD PHONEME OCCURRING FOR THE REMAINING VECTORS OF SPEECH IS COMPUTED.

```

*****}

```

```

PROCEDURE WEQ1SGE3;

```

```

VAR

```

```

    VECTOR,N:INTEGER;

```

```

BEGIN (*WEQ1SGE3*)

```

```

    FOR N:=W TO (W+1) DO

```

```

        BEGIN (*FOR*)

```

```

            VECTOR:=S;

```

```

            FINDMAX (N,VECTOR,PR,ERROR);

```

```

            PHSCR[N]:=PR;

```

```

            TERROR[N]:=ERROR;

```

```

        END (*FOR*);

```

```

        IF PHSCR[W]>PHSCR[W+1] THEN

```

```

            PWISMAX

```

```

        ELSE IF (PHSCR[W]<PHSCR[W+1]) THEN

```

```

            BEGIN (*ELSE*)

```

```

                PW:=PHSCR[W];

```

```

                PW1:=PHSCR[W+1];

```

```

IF (ENDWORD-S-NOT0AVG) >= 0 THEN
  NOAVG:=NOT0AVG
ELSE
  NOAVG:=3;
FOR VECTOR:=(S+1) TO (S+NOAVG) DO
  BEGIN (*FOR*)
    N:=W;
    FINDMAX (N,VECTOR,PR,ERROR);
    PW:=PW+PR;
    N:=W+1;
    FINDMAX (N,VECTOR,PR,ERROR);
    PW1:=PW1+PR;
  END (*FOR*);
  PW:=PW/(NOAVG+1);
  PW1:=PW1/(NOAVG+1);
  IF (THR1E*POWER(PW,THR1F))>PW1 THEN
    BEGIN(*IF*)
      PW1SMAX;
    END(*IF*)
  ELSE
    BEGIN (*ELSE*)
      PW11SMAX;
      WHILE (ENDWORD-S)>=0 DO
        BEGIN (*WHILE*)
          VECTOR:=S;
          FINDMAX(W,VECTOR,PR,ERROR);
          PHSCR[W]:=PR;
          TERROR[W]:=ERROR;
          PW1SMAX;
        END (*WHILE*);
        FLAG:=FALSE;
      END (*ELSE*);
    END (*ELSE*);
  END (*ELSE*);

```

```

END (*PROCEDURE WEQ1SGE3*);

```

```

{*****}

```

PROCEDURE WGE2SLE2 DESCRIPTION -----

THIS PROCEDURE IS CALLED WHEN MORE THAN 2 WORD PHONEMES
REMAIN AND LESS THAN 3 VECTORS OF SPEECH REMAIN. THIS PROCEDURE

ONLY USES INFORMATION CONCERNING THE CURRENT VECTOR TO DETERMINE
WHETHER OR NOT A TRANSITION OCCURS.

*****}

PROCEDURE WGE2SLE2;

VAR

N:INTEGER;

BEGIN (*WGE2SLE2*)

FOR N:=W TO (W+2) DO

BEGIN (*FOR*)

VECTOR:=S;

FINDMAX(N,VECTOR,PR,ERROR);

PHSCR[N]:=PR;

TERROR[N]:=ERROR;

END (*FOR*);

IF ((PHSCR[W]>PHSCR[W+1])AND(PHSCR[W]>PHSCR[W+2])) THEN

PWISMAX

ELSE IF ((PHSCR[W+1]>PHSCR[W])AND(PHSCR[W+1]>PHSCR[W+2])) THEN

PW1ISMAX

ELSE IF ((PHSCR[W+2]>PHSCR[W])AND(PHSCR[W+2]>PHSCR[W+1])) THEN

PW2ISMAX;

IF (ENDWORD-S)<0 THEN

BEGIN (*IF*)

WHILE (WORDLENGTH-W)>0 DO

BEGIN (*WHILE*)

N:=W;

WDELETED(N);

W:=W+1;

END (*WHILE*);

END (*IF*);

END (*PROCEDURE WGE2SLE2*);

{*****}

PROCEDURE COMPUTESCORE DESCRIPTION -----

THIS PROCEDURE COMPUTES THE TOTAL WORD SCORE BASED ON THE VALUES OF THE FOLLOWING VARIABLES: DELCNT, WORDLENGTH, DELW, PWORD, AND SCORECOUNT. REFER TO THE GLOBAL VARIABLE DESCRIPTIONS, PROVIDED AT THE BEGINNING OF THE MAIN PROGRAM, FOR AN EXPLANATION OF EACH OF THESE VARIABLES.

*****}

PROCEDURE COMPUTESCORE;

VAR

ARGUMENT:REAL;

BEGIN (*COMPUTESCORE*)

{Compute the total deletion score.}

IF DELCNT(>)0 THEN

BEGIN(*IF*)

ARGUMENT:=1-(((1-(DELCNT/WORDLENGTH))*DCNG)+(1-DCNG)*(DELW/DELCNT));

ARGUMENT:=ARGUMENT/DCNE;

WORDSCORE:=1-POWER(ARGUMENT,DCNF);

END(*IF*)

ELSE

BEGIN(*ELSE*)

WORDSCORE:=1;

END(*ELSE*);

{Compute the total word score.}

WORDSCORE:=(DELG*WORDSCORE)+(1-DELG)*(PWORD/SCORECOUNT);

END(*PROCEDURE COMPUTESCORE*);

{*****}

PROCEDURE FINDMAX DESCRIPTION

THIS PROCEDURE DETERMINES THE PLAUSIBILITY OF A GIVEN WORD PHONEME OCCURRING FOR A GIVEN VECTOR OF SPEECH. THE PLAUSIBILITY IS CALCULATED BASED ON WHETHER A SUBSTITUTION ERROR OR AN INSERTION ERROR OCCURS. A SUBSTITUTION ERROR IS SAID TO HAVE OCCURRED IF THE PHONEME BEING EXAMINED IS AMONG THE NUMBER OF GUESSES PROVIDED, IF NOT, AN INSERTION ERROR IS SAID TO HAVE OCCURRED. ONCE THE TYPE OF ERROR IS DECIDED, THEN A SCORE IS CALCULATED FOR EACH OF THE GUESSES, BASED ON THE VALUE ASSIGNED TO EACH GUESS BY THE ACOUSTIC PROCESSOR, AND ON THE STATISTICS COLLECTED FROM THE TRAINING SET. THE MAXIMUM OF THESE SCORES THEN BECOMES THE PLAUSIBILITY FOR THE WORD PHONEME BEING EXAMINED.

*****}

PROCEDURE FINDMAX;

VAR

VSCORE,SUBSCORE,AINSSCORE:REAL;
MAXSCORE:ARRAY[1..NOGUESSES] OF REAL;
MAX:REAL;
IMAX:INTEGER;
K:INTEGER;
I,J:INTEGER;
ARGUMENT:REAL;

BEGIN (*FINDMAX*)

WITH WSTATS DO

BEGIN (*WITH*)

K:=VECTOR-OFFSET;

{Determine if an insertion or substitution error
occurred for the word phoneme for the given vector.}

ERROR:='INSRT';

FOR I:=1 TO NOGUESSES DO

BEGIN (*FOR*)

IF (WDATA.PHREP[N]=CHPHONE[I,K]) THEN

ERROR:='SUBST';

IF (XSUBY[N,CHPHONE[I,K]]) (XSUBY[N,WDATA.PHREP[N]]*STHR) THEN

BEGIN(*IF*)

IF (XSUBY[N,CHPHONE[I,K]]>0) AND (N<(MAXPHONES)) THEN

```

BEGIN(*IF*)
  IF (CHPHONE[I,K](>)WDATA.PHREP[N+1])THEN
    BEGIN(*IF*)
      ERROR:='SUBST';
    END(*IF*);
  END(*IF*);
  IF N=MAXPHONES THEN
    BEGIN(*IF*)
      ERROR:='SUBST';
    END(*IF*);
  END(*IF*);
END (*FOR*);
IF ERROR='SUBST' THEN
  BEGIN (*IF*)

```

{Compute the vector score given that a substitution error occurred.}

```

FOR I:=1 TO NOGUESSES DO
  BEGIN (*FOR*)
    ARGUMENT:=SFACTOR[K]/SFE;
    VSCORE:=(1-CHVALUE[I,K]/100)/CHVE;
    VSCORE:=(1-POWER(VSCORE,CHVF))*(1-POWER(ARGUMENT,SFF));
    IF XSUBY[N,CHPHONE[I,K]](>)0 THEN
      BEGIN(*IF*)
        ARGUMENT:=(1-(XSUBY[N,CHPHONE[I,K]]/(NOSPOKE-NDEL[N])))/STATE;
        SUBSCORE:=1-POWER(ARGUMENT,STATF);
      END(*IF*)
    ELSE
      BEGIN(*ELSE*)
        SUBSCORE:=0;
      END(*ELSE*);
    ARGUMENT:=(((1-STATG)*VSCORE)+((STATG)*(SUBSCORE)))/SUBE;
    MAXSCORE[I]:=POWER(ARGUMENT,SUBF);
  END (*FOR*);
END (*IF*)
ELSE IF ERROR='INSRT' THEN
  BEGIN (*ELSE*)

```

{Compute the vector score given that an insertion error occurred.}

```

FOR I:=1 TO NOGUESSES DO
  BEGIN (*FOR*)
    ARGUMENT:=SFACTOR[K]/SFE;
    VSCORE:=(1-CHVALUE[I,K]/100)/CHVE;
    VSCORE:=(1-POWER(VSCORE,CHVF))*(1-POWER(ARGUMENT,SFF));
    IF XINAFTZ[N,CHPHONE[I,K]](>)0 THEN
      BEGIN(*IF*)
        ARGUMENT:=(1-(XINAFTZ[N,CHPHONE[I,K]]/NOSPOKE))/STATE;
        AINSSCORE:=1-POWER(ARGUMENT,STATF);
      END(*IF*)
    ELSE
      BEGIN(*ELSE*)

```

```

        AINSSCORE:=0;
        END(*ELSE*);
        ARGUMENT:=(((1-STATG)*VSCORE)+((STATG)*AINSSCORE))/INSE;
        MAXSCORE[I]:=POWER(ARGUMENT,INSF);
    END (*FOR*);
END (*ELSE*);

```

{Compute the overall vector score which is equal to the average vector score for all the choices given for the vector.}

```

PR:=0;
FOR I:=1 TO NOGUESSES DO
    BEGIN (*FOR*)
        PR:=PR+MAXSCORE[I];
    END (*FOR*);
PR:=PR/NOGUESSES;

```

```

END(*WITH*);

```

```

END (*PROCEDURE FINDMAX*);

```

```

{*****}

```

PROCEDURE PWISMAX DESCRIPTION

THIS PROCEDURE IS CALLED WHEN IT IS DECIDED THAT THE CURRENT WORD PHONEME OCCURRED FOR THE CURRENT SPEECH VECTOR. THE PURPOSE OF THIS PROCEDURE IS TO STORE THE SCORE, PHONEME CHOSEN, AND THE ERROR THAT OCCURRED FOR THE GIVEN VECTOR; AND TO INCREMENT THE COUNT THAT INDICATES THE CURRENT VECTOR (S). IF THE VECTOR NUMBER IS ABOVE A CERTAIN VALUE, THE NEXT SPEECH RECORD IS RETRIEVED AND PLACED IN MEMORY.

```

{*****}

```

```

PROCEDURE PWISMAX;

```

```

VAR
    K:INTEGER;

```

```

BEGIN (*PWISMAX*)
    K:=S-OFFSET;
    SCORE[K]:=PHSCR[W];
    PHONE[K]:=WDATA.PHREP[W];
    VERROR[K]:=TERROR[W];
    SCORECOUNT:=SCORECOUNT+1;
    PWORD:=PWORD+PHSCR[W];
    IF VERROR[K]='INSRT' THEN
    BEGIN(*IF*)
        INSRTSCORE:=INSRTSCORE+SCORE[K];
    END(*IF*);

    S:=S+1;
    IF ((S-OFFSET)=70)AND(IOUT.CONTFLAG=1) THEN
    BEGIN (*IF*)
        GETRECORD;
    END (*IF*);
END (*PROCEDURE PWISMAX*);

```

{*****}

PROCEDURE PWIISMAX DESCRIPTION

THIS PROCEDURE IS CALLED WHEN A TRANSITION OCCURS BETWEEN THE CURRENT ORD PHONEME AND THE NEXT ONE . THE PURPOSE OF THIS PROCEDURE IS TO STORE THE SCORE, PHONEME CHOSEN, AND THE ERROR THAT OCCURRED FOR THE GIVEN VECTOR; AND TO INCREMENT THE COUNT THAT INDICATES THE CURRENT VECTOR (S). IF THE VECTOR NUMBER IS ABOVE A CERTAIN VALUE, THE NEXT SPEECH RECORD IS RETRIEVED AND PLACED IN MEMORY. A CHECK IS ALSO MADE TO DETERMINE IF A SUBSTITUTION ERROR OCCURED FOR THE CURRENT PHONEME, IF NOT, A DELETION ERROR IS SAID TO HAVE OCCURRED, AND A DELETION SCORE IS COMPUTED.

*****}

PROCEDURE PWIISMAX;

VAR
K,J:INTEGER;

```

BEGIN (*PW1ISMAX*)

  K:=S-OFFSET;
  J:=S-OFFSET-1;
  IF J=0 THEN
    BEGIN(*IF*)
      J:=1;
      PHONE[1]:=0;
      VERROR[1]:='XXXXX';
    END(*IF*);
    WHILE (VERROR[J]='INSRT') AND (J)(BEGINWORD+OFFSET)) DO
      BEGIN (*WHILE*)
        J:=J-1;
      END (*WHILE*);
      IF PHONE[J]<>WDATA.PHREP[W] THEN
        BEGIN(*IF*)
          WDELETED (W);
        END (*IF*);
        W:=W+1;
        PWORD:=PWORD+PHSCR[W];
        SCORECOUNT:=SCORECOUNT+1;
        SCORE[K]:=PHSCR[W];
        PHONE[K]:=WDATA.PHREP[W];
        VERROR[K]:=TERROR[W];
        IF VERROR[K]='INSRT' THEN
          BEGIN(*IF*)
            INSRTSCORE:=INSRTSCORE+SCORE[K];
          END(*IF*);
        END(*IF*);

      S:=S+1;
      IF ((S-OFFSET))=70)AND(1OUT.CONTFLAG=1) THEN
        BEGIN (*IF*)
          GETRECORD;
        END (*IF*);
      END (*PROCEDURE PW1ISMAX*);

```

(*****)

PROCEDURE PW2ISMAX DESCRIPTION

THIS PROCEDURE IS CALLED WHEN A TRANSITION OCCURS BETWEEN THE CURRENT WORD PHONEME AND THE PHONEME FOLLOWING THE NEXT ONE. THE PURPOSE OF THIS PROCEDURE IS TO STORE THE SCORE, WORD PHONEME CHOSEN, AND THE ERROR THAT OCCURRED FOR THE GIVEN VECTOR; AND TO

INCREMENT THE COUNT THAT INDICATES THE CURRENT VECTOR (S). IF THE VECTOR NUMBER IS ABOVE A CERTAIN VALUE, THE NEXT SPEECH RECORD IS RETRIEVED AND PLACED IN MEMORY. A CHECK IS ALSO MADE TO DETERMINE IF A SUBSTITUTION ERROR OCCURED FOR THE CURRENT PHONEME, IF NOT, A DELETION ERROR IS SAID TO HAVE OCCURRED, AND A DELETION SCORE IS COMPUTED. A DELETION SCORE IS ALSO COMPUTED FOR THE WORD PHONEME THAT WAS SKIPPED.

*****}

PROCEDURE PW2ISMAX;

VAR

K,J:INTEGER;

BEGIN (*PW2ISMAX*)

K:=S-OFFSET;

J:=S-OFFSET-1;

IF J=0 THEN

BEGIN(*IF*)

J:=1;

PHONEC[1]:=0;

VERROR[1]:='XXXXX';

END(*IF*);

WHILE (VERROR[J]='INSRT')AND(J)(BEGINWORD+OFFSET)) DO

BEGIN (*WHILE*)

J:=J-1;

END (*WHILE*);

IF PHONEC[J] (>) WDATA.PHREP[W] THEN

BEGIN (*IF*)

WDELETED (W);

END (*IF*);

W:=W+1;

WDELETED (W);

PWORD:=PWORD+PHSCREW+1;

SCORECOUNT:=SCORECOUNT+1;

W:=W+1;

SCOREC[K]:=PHSCREW;

PHONEC[K]:=WDATA.PHREP[W];

VERROR[K]:=TERROR[W];

IF VERROR[K]='INSRT' THEN

BEGIN(*IF*)

INSRTSCORE:=INSRTSCORE+SCOREC[K];

END(*IF*);

S:=S+1;

```

      IF ((S-OFFSET))=70)AND(IOUT.CONTFLAG=1) THEN
      BEGIN (*IF*)
        GETRECORD;
      END (*IF*);
    END (*PROCEDURE PW2ISMAX*);

```

{*****}

PROCEDURE WDELETED DESCRIPTION

THIS PROCEDURE COMPUTES A DELETION SCORE FOR A DELETED WORD
 PHONEME BASED ON THE NUMBER OF TIMES THE PHONEME WAS DELETED IN THE
 TRAINING SET.

{*****}

```

PROCEDURE WDELETED;

```

```

  VAR

```

```

    ARGUMENT:REAL;

```

```

  BEGIN (*WDELETED*)

```

```

    DELCNT:=DELCNT+1;

```

```

    ARGUMENT:=(WSTATS.NDEL[N]/WSTATS.NOSPOKE)/DELE;

```

```

    DELW:=POWER(ARGUMENT,DELF)+DELW;

```

```

  END (*PROCEDURE WDELETED*);

```

{*****}

PROCEDURE GETRECORD DESCRIPTION

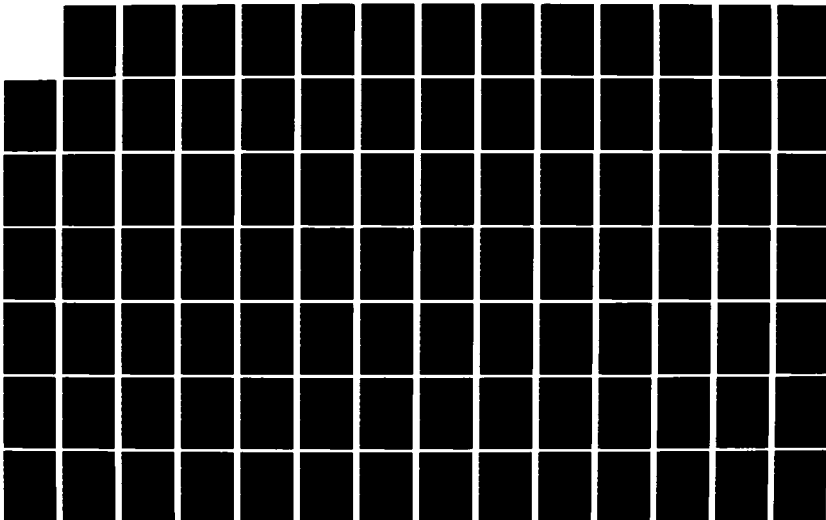
THIS PROCEDURE RETRIEVES THE NEXT RECORD OF THE SPEECH
 FILE, AND PLACES IT IN THE APROPRIATE ARRAYS.

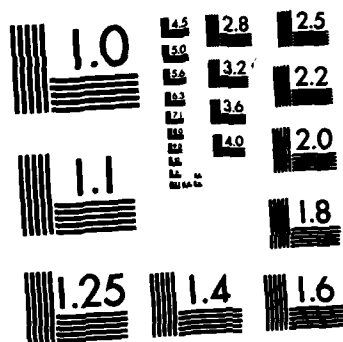
{*****}

AD-A124 851

ISOLATED WORD RECOGNITION USING FUZZY SET THEORY(U) AIR 3/4
FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING G J MONTGOMERY DEC 82 AFIT/GE/EE/82D-74
F/G 5/8 NL

UNCLASSIFIED





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

PROCEDURE GETRECORD;

VAR

J,I:INTEGER;

BEGIN (*GETRECORD*)

SEEK(SPECH,I RECORD);
GET(SPECH);
IOUT:=SPECH^;
OFFSET:=OFFSET+RCRDLENGTH;

{Assign first half of record array the values
stored in the second half of the record.}

FOR J:=1 TO RCRDLENGTH DO
BEGIN (*FOR*)
FOR I:=1 TO NOGUESSES DO
BEGIN (*FOR*)
CHPHONE[I,J]:=CHPHONE[I,J+RCRDLENGTH];
CHVALUE[I,J]:=CHVALUE[I,J+RCRDLENGTH];
END (*FOR*);
SFACTOR[J]:=SFACTOR[J+RCRDLENGTH];
IOUT.ISCORE[J]:=SCORE[J];
IOUT.IPHONE[J]:=PHONE[J];
IOUT.IERROR[J]:=ERROR[J];
SCORE[J]:=SCORE[J+RCRDLENGTH];
PHONE[J]:=PHONE[J+RCRDLENGTH];
ERROR[J]:=ERROR[J+RCRDLENGTH];
END (*FOR*);
SEEK(SPECH,I RECORD);
SPECH^:=IOUT;
PUT(SPECH);

{Assign second half of array the values of the
next record of the speech file.}

I RECORD:=I RECORD+1;
SEEK(SPECH,(I RECORD+1));
GET(SPECH);
IOUT:=SPECH^;
WITH IOUT DO
BEGIN (*WITH*)
FOR J:=1 TO RCRDLENGTH DO
BEGIN (*FOR*)
FOR I:=1 TO NOGUESSES DO
BEGIN (*FOR*)
CHPHONE[I,J+RCRDLENGTH]:=I CHPHONE[I,J];
CHVALUE[I,J+RCRDLENGTH]:=I CHVALUE[I,J];
END (*FOR*);
SFACTOR[J+RCRDLENGTH]:=I SFACTOR[J];
SCORE[J+RCRDLENGTH]:=I SCORE[J];
PHONE[J+RCRDLENGTH]:=I PHONE[J];

```

        VERROR[J+RCRDLENGTH]:=IERROR[J];
        IF ICHPHONE[1,J] (>) 0 THEN
        BEGIN (*IF*)
            ENDWORD:=J+OFFSET+RCRDLENGTH;
        END (*IF*);
    END (*FOR*);
END (*WITH*);

```

```

END (*PROCEDURE GETRECORD*);

```

```

BEGIN (*SCOREWORD*)
    WRITELN('SCOREWORD');

    IRECORD:=BEGINREC;

    {Initialize offset.}

    OFFSET:=0;
    WHILE (OFFSET+RCRDLENGTH)<BEGINWORD DO
    BEGIN(*WHILE*)
        OFFSET:=OFFSET+RCRDLENGTH;
        IRECORD:=IRECORD+1;
    END(*WHILE*);
    RESET(SPECH);
    GET2RECORDS;
    RESET(WFILE);
    RESET(SFILE);

    {Initialize all program variables.}

    TEMPWORD:='XXXXX';
    WORDNO:=0;
    WHILE (NOT EOF(WFILE)) AND (WORD<>TEMPWORD) DO
    BEGIN (*WHILE*)
        SEEK(WFILE,WORDNO);
        GET(WFILE);
        WDATA:=WFILE^;
        GET(WFILE);
        TEMPWORD:=WDATA.SPELLING;
        WORDNO:=WORDNO+1;
    END (*WHILE*);
    WORDNO:=WORDNO-1;
    WORDLENGTH:=0;
    FLAG:=FALSE;

```

```

REPEAT
  WORDLENGTH:=WORDLENGTH+1;
  IF WDATA.PHREP[WORDLENGTH]=0 THEN
    BEGIN(*IF*)
      FLAG:=TRUE;
      WORDLENGTH:=WORDLENGTH-1;
    END(*IF*);
  IF WORDLENGTH=MAXPHONES THEN
    FLAG:=TRUE;
  UNTIL FLAG;
  SEEK(SFILE,WORDNO);
  GET(SFILE);
  WSTATS:=SFILE^;
  IF WSTATS.NOSPOKE=0 THEN
    BEGIN(*IF*)
      WSTATS.NOSPOKE:=1;
    END(*IF*);
  S:=BEGINWORD;
  W:=1;
  PWORD:=0.0;
  INSRSCORE:=0.0;
  SCORECOUNT:=0;
  DELW:=0.0;
  DELCNT:=0;
  FOR I:=1 TO MAXVECTORS DO
    BEGIN(*FOR*)
      SCORE[I]:=0.0;
      PHONE[I]:=0;
      VERROR[I]:='XXXXX';
    END (*FOR*);

  {Determine word phoneme score and the error that
  occurs for each vector of the speech file based on the
  number of word phonemes and vectors remaining to be
  examined.}

  WHILE ((WORDLENGTH-W)=2) AND ((ENDWORD-S)=3) DO
    BEGIN (*WHILE*)
      WGE2SCE3;
    END (*WHILE*);
  WHILE ((ENDWORD-S)=0) DO
    BEGIN (*WHILE*)
      IF ((ENDWORD-S)<= 2)AND((WORDLENGTH-W) > 0) THEN
        BEGIN (*IF*)
          WHILE ((ENDWORD-S)=0) AND ((WORDLENGTH-W) > 0) DO
            BEGIN (*WHILE*)
              IF (WORDLENGTH-W)=2 THEN
                WGE2SLE2
              ELSE IF (WORDLENGTH-W)=1 THEN
                WEQ1SLE2;
            END (*WHILE*);
          END (*IF*)
        ELSE IF ((ENDWORD-S)=3)AND ((WORDLENGTH-W)=1) THEN

```

```

BEGIN (*ELSE*)
    WEQ1SGE3;
END(*ELSE*)
ELSE IF (WORDLENGTH-W)=0 THEN
BEGIN(*ELSE*)
    WHILE (ENDWORD-S)=0 DO
        BEGIN(*WHILE*)
            VECTOR:=S;
            FINDMAX(W,VECTOR,PR,ERROR);
            PHSCR[W]:=PR;
            TERROR[W]:=ERROR;
            PWISMAX;
        END (*WHILE*);
    END(*ELSE*)
    ELSE
    BEGIN(*ELSE*)
        WRITELN('ERROR1 IN PROCEDURE SCOREWORD.');
```

END(*ELSE*);

```

    END (*WHILE*);
    COMPUTESCORE;
    SEEK(SPECH,I RECORD);
    GET(SPECH);
    IOUT:=SPECH^;
    WITH IOUT DO
    BEGIN (*WITH*)
        FOR J:=1 TO RCRDLENGTH DO
        BEGIN (*FOR*)
            ISCORE[J]:=SCORE[J];
            IPHONE[J]:=PHONE[J];
            IERROR[J]:=VERROR[J];
        END (*FOR*);
        SEEK(SPECH,I RECORD);
        SPECH^:=IOUT;
        PUT(SPECH);
        IF IOUT.CONTFLAG=1 THEN
        BEGIN(*IF*)
            IRECORD:=IRECORD+1;
            SEEK(SPECH,I RECORD);
            GET(SPECH);
            IOUT:=SPECH^;
            FOR J:=1 TO RCRDLENGTH DO
            BEGIN (*FOR*)
                ISCORE[J]:=SCORE[J+RCRDLENGTH];
                IPHONE[J]:=PHONE[J+RCRDLENGTH];
                IERROR[J]:=VERROR[J+RCRDLENGTH];
            END (*FOR*);
            SEEK(SPECH,I RECORD);
            SPECH^:=IOUT;
            PUT(SPECH);
        END(*IF*);
    END (*WITH*);
END(*PROCEDURE SCOREWORD*);

```

{*****}

PROCEDURE UPDATE DESCRIPTION

THIS PROCEDURE UPDATES A GIVEN WORD'S STATISTICS BASED ON THE WORD PHONEME CHOSEN AND THE ERROR THAT OCCURRED FOR EACH VECTOR OF A SPEECH FILE, WHICH WAS DETERMINED BY PROCEDURE SCOREWORD. (NOTE THAT PROCEDURE SCOREWORD MUST BE EXECUTED BEFORE THIS PROCEDURE IS CALLED). SINCE, IN GENERAL, EACH WORD PHONEME WILL BE CHOSEN FOR MORE THAN ONE VECTOR, THE SUBSTITUTION AND INSERTION STATISTICS FOR EACH WORD PHONEME ARE COMPUTED BY USING THE MAXIMUM SUBSTITUTION AND INSERTION SCORES PER PHONEME, FOR ALL THE VECTORS ASSOCIATED WITH A GIVEN WORD PHONEME.

THE FOLLOWING VARIABLES MUST BE INITIALIZED PRIOR TO CALLING THIS PROCEDURE: WORDNO, BEGINREC, BEGINWORD.

*****}

SEGMENT PROCEDURE UPDATE;

VAR

SUB,INS:ARRAY[1..NOPHONEMES] OF REAL;
VSCORE,ARGUMENT:REAL;
I,TEMP,INT,J:INTEGER;
OFFSET,S,K:INTEGER;
TEMPFLAG,FLAG:BOOLEAN;

BEGIN(*UPDATE*)

WRITELN('UPDATE');

{Get word's data records.}

RESET(SFILE);
SEEK(SFILE,WORDNO);
GET(SFILE);
WSTATS:=SFILE^;
RESET(WFILE);
SEEK(WFILE,WORDNO);
GET(WFILE);
WDATA:=WFILE^;
RESET(SPECH);
IRECORD:=BEGINREC;
OFFSET:=0;
WHILE(OFFSET+RCRDLENGTH)<=BEGINWORD DO
BEGIN (*WHILE*)

```

    OFFSET:=OFFSET+RCRDLENGTH;
    IRECORD:=IRECORD+1;
END(*WHILE*);
SEEK(SPECH,IRECORD);
GET(SPECH);
IOUT:=SPECH^;
WITH WSTATS DO
BEGIN (*WITH*)
    S:=BEGINWORD;
    NOSPOKE:=NOSPOKE+1;

    {Determine the plausibility of each prototype phoneme
    being substituted or inserted for each word phoneme
    for the speech file being examined.}

    FOR INT:=1 TO WORDLENGTH DO
    BEGIN (*FOR*)
        FLAG:=FALSE;
        K:=S-OFFSET;
        FOR I:=1 TO NOPHONEMES DO
        BEGIN (*FOR*)
            SUB[I]:=0.0;
            INS[I]:=0.0;
        END (*FOR*);
        TEMPFLAG:=TRUE;
        WHILE (WDATA.PHREP[INT]=IOUT.IPHONE[K])AND(S<=ENDWORD) AND
            (TEMPFLAG=TRUE) DO
        BEGIN (*WHILE*)
            WITH IOUT DO
            BEGIN (*WITH*)
                IF IERROR[K]='SUBST' THEN
                BEGIN (*IF*)
                    FLAG:=TRUE;
                    FOR I:=1 TO NOGUESSES DO
                    BEGIN (*FOR*)
                        TEMP:=ICHPHONE[I,K];
                        ARGUMENT:=ISFACTOR[K]/SFE;
                        VSCORE:=(1-ICHVALUE[I,K]/100)/CHVE;
                        VSCORE:=(1-POWER(VSCORE,CHVF))*(1-POWER(ARGUMENT,SFF));
                        IF ((VSCORE))SUB[TEMP]) THEN
                        BEGIN (*IF*)
                            SUB[TEMP]:=VSCORE;
                        END (*IF*);
                    END(*FOR*);
                END(*IF*);
            ELSE IF IERROR[K]='INSRT' THEN
            BEGIN (*ELSE*)
                FOR I:=1 TO NOGUESSES DO
                BEGIN (*FOR*)
                    TEMP:=ICHPHONE[I,K];
                    ARGUMENT:=ISFACTOR[K]/SFE;
                    VSCORE:=(1-ICHVALUE[I,K]/100)/CHVE;
                    VSCORE:=(1-POWER(VSCORE,CHVF))*(1-POWER(ARGUMENT,SFF));
                    IF ((VSCORE))INS[TEMP]) THEN

```

```

        BEGIN (*IF*)
            INS[TEMP]:=VSCORE;
        END (*IF*);
    END (*FOR*);
END (*ELSE*);
IF (((S-OFFSET)=RCRDLENGTH)AND(CONTFLAG=1)) THEN
    BEGIN (*IF*)

        {Get next record of speech file.}

        GET(SPECH);
        IOUT:=SPECH^;
        OFFSET:=OFFSET+RCRDLENGTH;
    END(*IF*)
    ELSE IF (S-OFFSET)=RCRDLENGTH THEN
        BEGIN(*ELSE*)
            TEMPFLAG:=FALSE;
            S:=S-1;
        END(*ELSE*);
    END(*WITH*);
    S:=S+1;
    K:=S-OFFSET;
END (*WHILE*);

{Test to determine if a deletion error occurred
for the word phoneme being evaluated.}

IF FLAG=FALSE THEN
    BEGIN (*IF*)

        {Update deletion statistic for word phoneme.}

        NDEL[INT]:=NDEL[INT]+1;
    END (*IF*);

    {Update word phoneme's insertion and
    substitution statistics.}

    FOR J:=1 TO NOPHONEMES DO
        BEGIN (*FOR*)
            XSUBY[INT,J]:=XSUBY[INT,J]+SUB[J];
            XINAFTZ[INT,J]:=XINAFTZ[INT,J]+INS[J];
        END(*FOR*);
    END (*FOR*);
END (*WITH*);

    SEEK (SFILE,WORDNO);
    SFILE^:=WSTATS;
    PUT(SFILE);

END (*PROCEDURE UPDATE*);

```

{*****}

PROCEDURE ISORECOG DESCRIPTION

THIS PROCEDURE ACCEPTS THE NAME OF A SPEECH FILE TO BE RECOGNIZED, AND COMPUTES A SCORE FOR EACH WORD IN THE VOCABULARY BASED ON HOW WELL THE WORD MATCHES THE GIVEN FILE. THE WORD WITH THE HIGHEST SCORE IS CHOSEN AS THE WORD THAT WAS SPOKEN. THE SPEECH FILE IS LIMITED TO ONE WORD SINCE THIS IS AN ISOLATED WORD RECOGNITION ALGORITHM. ONCE THE RECOGNITION SCORES ARE OBTAINED, THE WORD'S RECOGNITION STATISTICS ARE UPDATED (SINCE THE ACTUAL WORD SPOKEN IS STORED IN THE SPEECH FILE, THERE IS NO PROBLEM WITH UPDATING THE WRONG STATISTICS). THE RESULTS ARE STORED IN THE FILE SPECIFIED BY THE USER. THE USER ALSO HAS THE FOLLOWING TWO OPTIONS:

1. STORE THE SPEECH FILE IN FILE RECOG TO BE USED BY AN AUTOMATED PROCEDURE; AND
2. OUTPUT THE ERRORS, PHONEMES CHOSEN, AND THE SCORES OF THE VECTORS IN THE SPEECH FILE (OBTAINED BY PROCEDURE SCOREWORD) FOR EACH WORD IN THE VOCABULARY; AND OUTPUT THE FINAL VALUES OF PWORD, SCORECOUNT, DELCNT, DELW, AND WORDSCORE FOR EACH WORD.

*****}

SEGMENT PROCEDURE ISORECOG;

VAR

FILENAME:STRING[20];
WNUMBER,NOWORD:INTEGER;
RESULTS:TEXT;
FLAG:BOOLEAN;
CRECORD:INTEGER;
INT,I,J,LOFFSET:INTEGER;
TEMPWORD:STRING[SPELLENGTH];
ANSWER:STRING[3];
NAME:STRING[20];
OUTERRORS:STRING[3];
HIGHSCORE:REAL;
HIGHWORD:STRING[20];
NEXTHIGHEST:REAL;

BEGIN (*ISORECOG*)

INITFILE;
ENDFLAG:=FALSE;

```

WRITELN('DO YOU WANT TO STORE THE INPUT SPEECH FILE FOR');
WRITELN('FUTURE USE BY AN AUTOMATED PROCEDURE? IF YOU DO');
WRITELN('ENTER "YES"; ELSE ENTER "NO".');
WRITE('ANSWER = ');
READLN(ANSWER);
WRITELN;
IF ANSWER <> 'YES' THEN
BEGIN(*IF*)
    FILETITLE(SPECH,'TEMPFILE');
    REWRITE(SPECH);
END(*IF*)
ELSE
BEGIN(*ELSE*)
    FILENAME:=CONCAT(SPEAKER,'RECOG');
    FILETITLE(SPECH,FILENAME);
END(*ELSE*);

RESET(SPECH);
WRITELN;

```

{Get name of speech file, and assure that word
is in the current vocabulary.}

```

WRITELN('INPUT NAME OF SPEECHFILE');
WRITE('FILENAME = ');
READLN(NAME);
WRITELN;
FILETITLE(OUT2,NAME);
RESET (OUT2);
WITH IOUT DO
BEGIN (*WITH*)
    READLN(OUT2,SPELL);
    FOR I:= (LENGTH(SPELL)) TO (SPELLENGTH-1) DO
    BEGIN (*FOR*)
        SPELL:=CONCAT(SPELL,' ');
    END (*FOR*);
    REPEAT
        RESET (WFILE);
        FLAG:=FALSE;
        TEMPWORD:=' ';
        NOWORD:=0;
        WHILE (NOT EOF(WFILE))AND(SPELL<>TEMPWORD) DO
        BEGIN (*WHILE*)
            SEEK(WFILE,NOWORD);
            GET(WFILE);
            WDATA:=WFILE^;
            GET(WFILE);
            TEMPWORD:=WDATA.SPELLING;
            NOWORD:=NOWORD+ 1
        END (*WHILE*);
        WNUMBER:=NOWORD-1;
        IF (SPELL<>TEMPWORD) THEN
        BEGIN (*IF*)
            WRITELN;

```

```

        WRITELN ('NONE OF THE WORDS IN THE VOCABULARY');
        WRITELN ('AGREE WITH THE SPELLING GIVEN IN THE');
        WRITE ('FILE "',NAME,'" THE SPELLING IN ',NAME,' IS');
        WRITELN (' ',SPELL,' ');
        WRITELN ('DO YOU WANT TO CHANGE THE "',NAME,'" SPELLING?');
        WRITELN ('ENTER "YES" TO CHANGE, ELSE ENTER "NO"');
        WRITE ('ANSWER = ');
        READLN(ANSWER);
        IF ANSWER() 'YES' THEN
            BEGIN (*IF*)
                FLAG:=TRUE;
                ENDFLAG:=TRUE;
            END (*IF*)
        ELSE
            BEGIN (*ELSE*)
                WRITELN;
                WRITELN ('ENTER THE CORRECT SPELLING OF WORD. ');
                WRITE ('WORD = ');
                READLN (SPELL);
                WRITELN;
                FOR I:= (LENGTH(SPELL)) TO (SPELLENGTH-1) DO
                    BEGIN (*FOR*)
                        SPELL:=CONCAT(SPELL,' ');
                    END (*FOR*);
                END (*ELSE*);
            END (*IF*)
        ELSE
            BEGIN (*ELSE*)
                FLAG:=TRUE;
            END(*ELSE*);
        UNTIL FLAG;
    END (*WITH*);
    IF ENDFLAG=FALSE THEN
        BEGIN (*IF*)
            HIGHSCORE:=0.0;
            HIGHWORD:='XXXXX';
            NEXTHIGHEST:=0.0;

            {Store speech file in either a temporary file or
            in file RECOG in the format required by procedure
            SCOREWORD.}

            STOREWORD;
            WRITELN('INPUT NAME OF OUTPUT FILE');
            WRITE('FILENAME = ');
            READLN(FILENAME);
            WRITELN;

            {Output heading and recognition information to
            output file.}

            FILETITLE(RESULTS,FILENAME);
            REWRITE(RESULTS);
            WRITELN(RESULTS);

```

```

WRITELN(RESULTS);
WRITELN(RESULTS);
WRITELN(RESULTS);
WRITELN(RESULTS);
WRITELN(RESULTS, ' :20, WORD RECOGNITION RESULTS');
WRITELN(RESULTS);
WRITELN(RESULTS);
WRITELN(RESULTS, 'ACTUAL WORD = ', IOUT.SPELL);
WRITELN(RESULTS);
WRITELN(RESULTS, 'SPEECH FILENAME = ', NAME);
WRITELN(RESULTS);
WRITELN(RESULTS);

WRITELN('DO YOU WANT TO OUTPUT THE ERROR STATISTICS?');
WRITELN('ENTER "YES" TO OUTPUT THEM; ELSE ENTER "NO".');
WRITE('ANSWER = ');
READLN(OUTERRORS);
WRITELN;
RESET(WFILE);
NOWORD:=0;
WHILE NOT EOF(WFILE) DO
BEGIN(*WHILE*)
    SEEK(WFILE, NOWORD);
    GET(WFILE);
    WDATA:=WFILE^;
    WORD:=WDATA.SPELLING;
    WRITELN(RESULTS);
    WRITELN(RESULTS);
    WRITELN(RESULTS);
    WRITELN(RESULTS);
    WRITELN;
    WRITELN('-----');
    WRITELN('WORD ATTEMPTED = ', WORD);
    WRITELN(RESULTS, 'WORD ATTEMPTED = ', WORD);
    WRITELN(RESULTS, '-----');
    WRITELN(RESULTS);
    GETWORDVARS;
    WRITELN(RESULTS, 'THE WORD FUZZY VARIABLES THAT WERE USED FOLLOW');
    WRITELN(RESULTS);
    WRITELN(RESULTS, 'STHR = ', STHR, ' SUBE = ', SUBE, ' SUBF = ', SUBF);
    WRITELN(RESULTS, 'INSE = ', INSE, ' INSF = ', INSF);
    WRITELN(RESULTS, 'DELE = ', DELE, ' DELF = ', DELF, ' DELG = ', DELG);
    WRITELN(RESULTS, 'DCNE = ', DCNE, ' DCNF = ', DCNF, ' DCNG = ', DCNG);
    WRITELN(RESULTS, 'SFE = ', SFE, ' SFF = ', SFF);
    WRITELN(RESULTS, 'CHVE = ', CHVE, ' CHVF = ', CHVF);
    WRITELN(RESULTS, 'STATE = ', STATE, ' STATF = ', STATF, ' STATG = ', STATG);
    WRITELN(RESULTS, 'THR1E = ', THR1E, ' THR1F = ', THR1F);
    WRITELN(RESULTS, 'THR2E = ', THR2E, ' THR2F = ', THR2F);
    WRITELN(RESULTS);
    SCOREWORD;
    IF NOWORD=WNUMBER THEN
    BEGIN(*IF*)
        SEEK(SFILE, WNUMBER);
        GET(SFILE);

```

```

WSTATS:=SFILE^;
WSTATS.TOTALSCORE:=WSTATS.TOTALSCORE+WORDSCORE;
SEEK(SFILE,WNUMBER);
SFILE^:=WSTATS;
PUT(SFILE);
END(*IF*);
Writeln('WORD SCORE = ',WORDSCORE);
Writeln('-----');
Writeln(RESULTS,'WORD SCORE = ',WORDSCORE);
Writeln(RESULTS);
IF OUTERRORS='YES' THEN
BEGIN(*IF*)
    Writeln(RESULTS,'PWORD = ',PWORD,' SCORECOUNT = ',SCORECOUNT);
    Writeln(RESULTS,'DELW = ',DELW,' DELCNT = ',DELCNT);
END(*IF*);

{Determine highest scoring word, and score
for the next highest scoring word.}

IF (WORDSCORE<=HIGHSCORE)AND(WORDSCORE>NEXTHIGHEST) THEN
BEGIN(*IF*)
    NEXTHIGHEST:=WORDSCORE;
END(*IF*);
IF ((WORDSCORE)>HIGHSCORE) THEN
BEGIN(*IF*)
    NEXTHIGHEST:=HIGHSCORE;
    HIGHSCORE:=WORDSCORE;
    HIGHWORD:=WORD;
END(*IF*);
CRECORD:=BEGINREC;
FLAG:=TRUE;
IOFFSET:=0;
WHILE ((FLAG=TRUE)OR(IOUT.CONTFLAG=1))DO
BEGIN (*WHILE*)
    FLAG:=FALSE;
    SEEK(SPECH,CRECORD);
    GET(SPECH);
    IOUT:=SPECH^;
    CRECORD:=CRECORD+1;
    I:=0;
    IF OUTERRORS='YES' THEN
    BEGIN(*IF*)

        {Output vector error and recognition
        information to output file.}

        WITH IOUT DO
        BEGIN (*WITH*)
            WHILE I<RCRDLENGTH DO
            BEGIN (*WHILE*)
                Writeln(RESULTS);
                WRITE(RESULTS,'VECTOR          = ');
                FOR J:=1 TO 10 DO
                BEGIN(*FOR*)

```

```

        INT:=J+I+IOFFSET;
        WRITE(RESULTS,INT:9);
    END (*FOR*);
    WRITELN(RESULTS);
    WRITE(RESULTS,'PHONEME CHOSEN = ');
    FOR J:=1 TO 10 DO
    BEGIN(*FOR*)
        WRITE(RESULTS,IPHONE[J+1]:9);
    END(*FOR*);
    WRITELN(RESULTS);
    WRITE(RESULTS,'SCORE           = ');
    FOR J:=1 TO 10 DO
    BEGIN(*FOR*)
        WRITE(RESULTS,ISCORE[J+1]:9);
    END(*FOR*);
    WRITELN(RESULTS);
    WRITE(RESULTS,'ERROR           = ');
    FOR J:=1 TO 10 DO
    BEGIN(*FOR*)
        WRITE(RESULTS,IVERROR[J+1],':4);
    END(*FOR*);
    WRITELN(RESULTS);
    WRITELN(RESULTS);
    WRITELN(RESULTS);
    I:=I+10;
    END (*WHILE*);
    END (*WITH*);
    END(*IF*);
    IOFFSET:=IOFFSET+RCRDLENGTH;
    END (*WHILE*);
    SEEK(WFILE,NOWORD);
    GET(WFILE);
    GET(WFILE);
    NOWORD:=NOWORD+1;
    END (*WHILE*);

    {Update word's recognition statistics.}

    SEEK(WFILE,WNUMBER);
    GET(WFILE);
    WDATA:=WFILE^;
    SEEK(SFILE,WNUMBER);
    GET(SFILE);
    WSTATS:=SFILE^;
    WSTATS.NOATTEMPT:=WSTATS.NOATTEMPT+1;
    IF HIGHWORD=WDATA.SPELLING THEN
    BEGIN(*IF*)
        WSTATS.NOCORRECT:=WSTATS.NOCORRECT+1;
        WSTATS.TOTDIFF:=WSTATS.TOTDIFF+(HIGHSCORE-NEXTHIGHEST);
        IF (HIGHSCORE-NEXTHIGHEST)<WSTATS.MINDIFF THEN
        BEGIN(*IF*)
            WSTATS.MINDIFF:=HIGHSCORE-NEXTHIGHEST;
        END (*IF*);
    END(*IF*);

```

```
SEEK(SFILE,WNUMBER);  
SFILE^:=WSTATS;  
PUT(SFILE);  
WRITELN;  
WRITELN;  
WRITELN('THE RECOGNITION RESULTS ARE IN FILE "',FILENAME,'"');  
  
END(*IF*);  
CLOSE(RESULTS);  
  
END (*PROCEDURE ISORECOG*);
```

{*****}

PROCEDURE CHNGEREP DESCRIPTION

THIS PROCEDURE PROVIDES THE USER WITH THE ABILITY TO MANUALLY CHANGE THE PHONEME REPRESENTATION OF A SPECIFIED WORD. AFTER THE NEW REPRESENTATION IS ENTERED, THIS PROCEDURE WILL RE-INITIALIZE THE WORD'S STATISTICS USING THE WORD'S TRAINING SET STORED IN FILE SPECH. IF THE USER DOES NOT WANT THE STATISTICS TO BE INITIALIZED (THE USER MAY WANT TO ALTER THE REPRESENTATION OF SEVERAL WORDS, AND THEN EXECUTE THE AUTOMATED INITSTAT ROUTINE) THIS PROCEDURE CAN BE ABORTED WHEN "INITSTAT" APPEARS ON THE TERMINAL SCREEN.

*****}

SEGMENT PROCEDURE CHNGEREP;

VAR

W:INTEGER;
I,J:INTEGER;
NOWORD:INTEGER;
TEMPWORD:STRING[SPELLENGTH];
ANS:INTEGER;
FLAG:BOOLEAN;

BEGIN(*INIT1*)

INITFILE;

ENDFLAG:=FALSE;
WRITELN;

{Get word to be operated on, determine if it is in the current vocabulary, and set variable NOWORD to indicate the word.}

WRITELN ('INPUT EXACT SPELLING OF WORD WHOSE REPRESENTATION ');
WRITELN('IS TO BE CHANGED.');

WRITE ('WORD = ');

READLN (WORD);

FOR I:=(LENGTH(WORD)) TO (SPELLENGTH-1) DO

BEGIN(*FOR*)

WORD:=CONCAT(WORD,' ');

END(*FOR*);

WRITELN ;

RESET(SFILE);

```

RESET (WFILE);
TEMPWORD:=' ';
NOWORD:=0;
WHILE (NOT EOF(WFILE))AND(WORD<>TEMPWORD) DO
BEGIN (*WHILE*)
    SEEK(WFILE,NOWORD);
    GET(WFILE);
    WDATA:=WFILE^;
    GET(WFILE);
    TEMPWORD:=WDATA.SPELLING;
    NOWORD:=NOWORD+ 1
END (*WHILE*);
NOWORD:=NOWORD-1;
IF (WORD<>TEMPWORD) THEN
BEGIN (*IF*)
    WRITELN;
    WRITELN ('WORD DOES NOT EXIST. ');
    ENDFLAG:=TRUE;
END (*IF*)
ELSE
BEGIN (*ELSE*)

```

{Output word's current phoneme representation, and
get the new one.}

```

WITH WDATA DO
BEGIN (*WITH*)
    SPELLING:=WORD;
    WRITELN;
    WRITELN('THE CURRENT PHONEME REPRESENTATION FOLLOWS:');
    FOR I:=1 TO SPELLENGTH DO
    BEGIN(*FOR*)
        WRITE(WDATA.PHREPII:3);
    END(*FOR*);
    WRITELN;
    WRITELN;
    WRITELN;
    WRITELN ('ENTER THE NUMERIC VALUE (01 TO 71) FOR ');
    WRITELN ('EACH PHONEME IN THE WORD, UP TO A MAXIMUM OF ');
    WRITE (MAXPHONES);
    WRITELN (' PHONEMES. AFTER ALL WORD PHONEMES HAVE BEEN ');
    WRITELN ('ENTERED, ENTER "00". ');
    W:=1;
    ANS:=71;
    WHILE (W<=MAXPHONES)AND(ANS<>00) DO
    BEGIN (*WHILE*)
        WRITELN;
        WRITE ('WORD PHONEME ',W:2,' = ');
        READLN (ANS);
        IF (00<=ANS)AND(ANS<=71) THEN
        BEGIN (*IF*)
            PHREPI[W]:=ANS;
            W:=W+1
        END (*IF*)
    END (*WHILE*)

```

```

ELSE
  BEGIN (*ELSE*)
    WRITELN;
    WRITELN ('PHONEME NOT VALID, REENTER');
  END (*ELSE*);
END (*WHILE*);

```

```

IF PHREP[1] = 00 THEN
  BEGIN(*IF*)
    WRITELN('NO DATA RECORDED (WORD PHONEME 1 = 0).');
    ENDFLAG:=TRUE
  END(*IF*)

```

```

ELSE
  BEGIN (*ELSE*)
    IF W<MAXPHONES THEN
      BEGIN (*IF*)
        FOR J:= W TO MAXPHONES DO
          BEGIN (*FOR*)
            PHREP[J]:=00
          END (*FOR*);
        END (*IF*);
        SEEK(WFILE,NOWORD);
        WFILE^:=WDATA;
        PUT (WFILE);

```

{Initialize the word's statistics using the overall fuzzy variables and the word's training files stored in file SPECH.}

```

      GETFUZZYVARS;
      INITSTAT(NOWORD);
    END (*ELSE*);
  END (*WITH*);
END (*ELSE*);

```

```

END(*PROCEDURE CHNGEREP*);

```

{*****}

PROCEDURE CHNGEFUZ DESCRIPTION

THIS PROCEDURE ENABLES THE USER TO CHANGE EITHER THE OVERALL
FUZZY VARIABLES STORED IN FILE FUZZYVAR, OR A SPECIFIED WORD'S
FUZZY VARIABLES.

*****}

SEGMENT PROCEDURE CHNGEFUZ;

VAR

TEMPWORD:STRING[SPELLENGTH];
ANSWER:STRING[3];

PROCEDURE CHWORVAR;

VAR

I:INTEGER;

BEGIN(*CHWORVAR*)

INITFILE;
WRITELN;
WRITELN ('INPUT EXACT SPELLING OF WORD WHOSE VARIABLES ');
WRITELN('ARE TO BE CHANGED.');

WRITE ('WORD = ');
READLN (WORD);
FOR I:=(LENGTH(WORD)) TO (SPELLENGTH-1) DO
BEGIN(*FOR*)
WORD:=CONCAT(WORD,' ');
END(*FOR*);
WRITELN ;
RESET(SFILE);
RESET (WFILE);
TEMPWORD:=' ' ;
WORDNO:=0;
WHILE (NOT EOF(WFILE))AND(WORD<>TEMPWORD) DO
BEGIN (*WHILE*)
SEEK(WFILE,WORDNO);
GET(WFILE);

```

WDATA:=WFILE^;
GET(WFILE);
TEMPWORD:=WDATA.SPELLING;
WORDNO:=WORDNO+ 1
END (*WHILE*);
WORDNO:=WORDNO-1;
IF (WORD<>TEMPWORD) THEN
BEGIN (*IF*)
    WRITELN;
    WRITELN('WORD DOES NOT EXIST. ');
END (*IF*)
ELSE
BEGIN (*ELSE*)
    WITH WDATA DO
    BEGIN(*WITH*)
        REPEAT
            WRITELN;
            WRITELN('THE CURRENT FUZZY VARIABLES FOLLOW: ');
            WRITELN;
            WRITELN('STHR = ',WSTHR,' SUBE = ',WSUBE,' SUBF = ',WSUBF);
            WRITELN('INSE = ',WINSE,' INSF = ',WINSF);
            WRITELN('DELE = ',WDELE,' DELF = ',WDELF,' DELG = ',WDELG);
            WRITELN('DCNE = ',WDCNE,' DCNF = ',WDCNF,' DCNG = ',WDCNG);
            WRITELN('SFE = ',WSFE,' SFF = ',WSFF);
            WRITELN('CHVE = ',WCHVE,' CHVF = ',WCHVF);
            WRITELN('STATE= ',WSTATE,' STATF= ',WSTATF,' STATG= ',WSTATG);
            WRITELN('THR1E= ',WTHR1E,' THR1F= ',WTHR1F);
            WRITELN('THR2E= ',WTHR2E,' THR2F= ',WTHR2F);
            WRITELN;
            WRITELN('DO YOU WISH TO CHANGE ANY FUZZY VARIABLES? ');
            WRITELN('ENTER "YES" TO CHANGE; ELSE ENTER "NO".');
            WRITE('ANSWER = ');
            READLN(ANSWER);
            WRITELN;
            IF ANSWER= 'YES' THEN
            BEGIN(*IF*)
                WRITE('OLD STHR = ',WSTHR,' NEW STHR = ');
                READLN(WSTHR);
                WRITE('OLD SUBE = ',WSUBE,' NEW SUBE = ');
                READLN(WSUBE);
                WRITE('OLD SUBF = ',WSUBF,' NEW SUBF = ');
                READLN(WSUBF);
                WRITE('OLD INSE = ',WINSE,' NEW INSE = ');
                READLN(WINSE);
                WRITE('OLD INSF = ',WINSF,' NEW INSF = ');
                READLN(WINSF);
                WRITE('OLD DELE = ',WDELE,' NEW DELE = ');
                READLN(WDELE);
                WRITE('OLD DELF = ',WDELF,' NEW DELF = ');
                READLN(WDELF);
                WRITE('OLD DELG = ',WDELG,' NEW DELG = ');
                READLN(WDELG);
                WRITE('OLD DCNE = ',WDCNE,' NEW DCNE = ');
                READLN(WDCNE);
            END(*IF*)
        UNTIL ANSWER= 'NO';
    END(*WITH*)
END(*ELSE*)

```

```

WRITE('OLD DCNF = ',WDCNF,' NEW DCNF = ');
READLN(WDCNF);
WRITE('OLD DCNG = ',WDCNG,' NEW DCNG = ');
READLN(WDCNG);
WRITE('OLD SFE = ',WSFE,' NEW SFE = ');
READLN(WSFE);
WRITE('OLD SFF = ',WSFF,' NEW SFF = ');
READLN(WSFF);
WRITE('OLD CHVE = ',WCHVE,' NEW CHVE = ');
READLN(WCHVE);
WRITE('OLD CHVF = ',WCHVF,' NEW CHVF = ');
READLN(WCHVF);
WRITE('OLD STATE= ',WSTATE,' NEW STATE= ');
READLN(WSTATE);
WRITE('OLD STATF= ',WSTATF,' NEW STATF= ');
READLN(WSTATF);
WRITE('OLD STATG= ',WSTATG,' NEW STATG= ');
READLN(WSTATG);
WRITE('OLD THR1E= ',WTHR1E,' NEW THR1E= ');
READLN(WTHR1E);
WRITE('OLD THR1F= ',WTHR1F,' NEW THR1F= ');
READLN(WTHR1F);
WRITE('OLD THR2E= ',WTHR2E,' NEW THR2E= ');
READLN(WTHR2E);
WRITE('OLD THR2F= ',WTHR2F,' NEW THR2F= ');
READLN(WTHR2F);
WRITELN;
END(*IF*);

UNTIL (ANSWER<>'YES');

SEEK(WFILE,WORDNO);
WFILE^:=WDATA;
PUT(WFILE);
END(*WITH*);
END(*ELSE*);

END(*PROCEDURE CHWORVAR*);

```

```

BEGIN(*CHNGEFUZ*);

```

```

WRITELN('DO YOU WANT TO CHANGE A WORD'S FUZZY VARIABLES,');
WRITELN('OR THE OVERALL FUZZY VARIABLES?. ENTER "WORD" TO');
WRITELN('CHANGE A WORD'S VARIABLES. ');
WRITE('ANSWER = ');
READLN(ANSWER);
WRITELN;
IF ANSWER<>'WOR' THEN
BEGIN(*IF*)
  RESET(FUZZYVAR);

```

```

IF NOT EOF(FUZZYVAR) THEN
BEGIN(*IF*)
    GETFUZZYVARS;
END(*IF*)
ELSE
BEGIN(*ELSE*);
    STHR:=1;
    SUBE:=1;
    SUBF:=1;
    INSE:=1;
    INSF:=1;
    DELE:=1;
    DELF:=1;
    DELG:=1;
    DCNE:=1;
    DCNF:=1;
    DCNG:=1;
    SFE:=1;
    SFF:=1;
    CHVE:=1;
    CHVF:=1;
    STATE:=1;
    STATF:=1;
    STATG:=1;
    THR1E:=1;
    THR1F:=1;
    THR2E:=1;
    THR2F:=1;
END(*ELSE*);
REPEAT
    WRITELN;
    WRITELN('THE CURRENT FUZZY VARIABLES FOLLOW: ');
    WRITELN;
    WRITELN('STHR = ',STHR,' SUBE = ',SUBE,' SUBF = ',SUBF);
    WRITELN('INSE = ',INSE,' INSF = ',INSF);
    WRITELN('DELE = ',DELE,' DELF = ',DELF,' DELG = ',DELG);
    WRITELN('DCNE = ',DCNE,' DCNF = ',DCNF,' DCNG = ',DCNG);
    WRITELN('SFE = ',SFE,' SFF = ',SFF);
    WRITELN('CHVE = ',CHVE,' CHVF = ',CHVF);
    WRITELN('STATE= ',STATE,' STATF= ',STATF,' STATG= ',STATG);
    WRITELN('THR1E= ',THR1E,' THR1F= ',THR1F);
    WRITELN('THR2E= ',THR2E,' THR2F= ',THR2F);
    WRITELN;
    WRITELN('DO YOU WISH TO CHANGE ANY FUZZY VARIABLES? ');
    WRITELN('ENTER "YES" TO CHANGE; ELSE ENTER "NO".');
    WRITE('ANSWER = ');
    READLN(ANSWER);
    WRITELN;
    IF ANSWER= 'YES' THEN
    BEGIN(*IF*)
        WRITE('OLD STHR = ',STHR,' NEW STHR = ');
        READLN(STHR);
        WRITE('OLD SUBE = ',SUBE,' NEW SUBE = ');
        READLN(SUBE);
    END(*IF*);

```

```

WRITE('OLD SUBF = ',SUBF,' NEW SUBF = ');
READLN(SUBF);
WRITE('OLD INSE = ',INSE,' NEW INSE = ');
READLN(INSE);
WRITE('OLD INSF = ',INSF,' NEW INSF = ');
READLN(INSF);
WRITE('OLD DELE = ',DELE,' NEW DELE = ');
READLN(DELE);
WRITE('OLD DELF = ',DELF,' NEW DELF = ');
READLN(DELF);
WRITE('OLD DELG = ',DELG,' NEW DELG = ');
READLN(DELG);
WRITE('OLD DCNE = ',DCNE,' NEW DCNE = ');
READLN(DCNE);
WRITE('OLD DCNF = ',DCNF,' NEW DCNF = ');
READLN(DCNF);
WRITE('OLD DCNG = ',DCNG,' NEW DCNG = ');
READLN(DCNG);
WRITE('OLD SFE = ',SFE,' NEW SFE = ');
READLN(SFE);
WRITE('OLD SFF = ',SFF,' NEW SFF = ');
READLN(SFF);
WRITE('OLD CHVE = ',CHVE,' NEW CHVE = ');
READLN(CHVE);
WRITE('OLD CHVF = ',CHVF,' NEW CHVF = ');
READLN(CHVF);
WRITE('OLD STATE= ',STATE,' NEW STATE= ');
READLN(STATE);
WRITE('OLD STATF= ',STATF,' NEW STATF= ');
READLN(STATF);
WRITE('OLD STATG= ',STATG,' NEW STATG= ');
READLN(STATG);
WRITE('OLD THR1E= ',THR1E,' NEW THR1E= ');
READLN(THR1E);
WRITE('OLD THR1F= ',THR1F,' NEW THR1F= ');
READLN(THR1F);
WRITE('OLD THR2E= ',THR2E,' NEW THR2E= ');
READLN(THR2E);
WRITE('OLD THR2F= ',THR2F,' NEW THR2F= ');
READLN(THR2F);
WRITELN;
END(*IF*);

UNTIL (ANSWER<>'YES');

```

```

REWRITE(FUZZYVAR);
WRITELN(FUZZYVAR,STHR,SUBE,SUBF);
WRITELN(FUZZYVAR,INSE,INSF);
WRITELN(FUZZYVAR,DELE,DELF,DELG);
WRITELN(FUZZYVAR,DCNE,DCNF,DCNG);
WRITELN(FUZZYVAR,SFE,SFF);
WRITELN(FUZZYVAR,CHVE,CHVF);
WRITELN(FUZZYVAR,STATE,STATF,STATG);

```

```
WRITELN(FUZZYVAR,THR1E,THR1F);
WRITELN(FUZZYVAR,THR2E,THR2F);
CLOSE(FUZZYVAR);
```

```
END(*IF*)
ELSE
BEGIN(*ELSE*)
    CHWORVAR;
END(*ELSE*);
```

```
END(*PROCEDURE CHNGEFUZ*);
```

```
{*****}
```

PROCEDURE CHNGEOPT DESCRIPTION

THIS PROCEDURE PROVIDES THE USER WITH THE ABILITY TO CHANGE EITHER THE INITIAL OPTIMUM FUZZY VARIABLE VALUES AND LIMITS STORED IN FILE OPTFUZZY, OR A SPECIFIED WORD'S OPTIMUM FUZZY VARIABLE VALUES AND LIMITS STORED IN FILE FZOPT.

```
*****}
```

```
SEGMENT PROCEDURE CHNGEOPT;
```

```
VAR
```

```
WNUMB,I,J:INTEGER;
FNAME:STRING[5];
ANSWER:STRING[3];
TEMPWORD:STRING[SPELLENGTH];
```

```
BEGIN(*CHNGEOPT*);
```

```
WRITELN('DO YOU WANT TO CHANGE A WORD'S FUZZY VARIABLES,');
WRITELN('OR THE OVERALL FUZZY VARIABLES?. ENTER "WORD" TO');
WRITELN('CHANGE A WORD'S VARIABLES. ');
WRITE('ANSWER = ');
```

```

READLN(ANSWER);
WRITELN;
IF ANSWER<>'WOR' THEN
BEGIN(*IF*)
  RESET(OPTFUZZY);
  IF NOT EOF(OPTFUZZY) THEN
  BEGIN(*IF*)
    WITH FLIMIT DO
    BEGIN(*WITH*)
      FOR I:=1 TO NOFUZZYVARS DO
      BEGIN(*FOR*)
        READLN(OPTFUZZY,FUZVARC[I],MAXLIMITC[I],MINLIMITC[I],FUZNAMEC[I]);
      END(*FOR*);
      RECScore:=0;
    END(*WITH*);
  END(*IF*);
  ELSE
  BEGIN(*ELSE*);
    WITH FLIMIT DO
    BEGIN(*WITH*)
      FUZNAME[1]:='STHR';
      FUZNAME[2]:='SUBE';
      FUZNAME[3]:='SUBF';
      FUZNAME[4]:='INSE';
      FUZNAME[5]:='INSP';
      FUZNAME[6]:='DELE';
      FUZNAME[7]:='DELF';
      FUZNAME[8]:='DELG';
      FUZNAME[9]:='DCNE';
      FUZNAME[10]:='DCNF';
      FUZNAME[11]:='DCNG';
      FUZNAME[12]:='SFE';
      FUZNAME[13]:='SFF';
      FUZNAME[14]:='CHVE';
      FUZNAME[15]:='CHVF';
      FUZNAME[16]:='STATE';
      FUZNAME[17]:='STATF';
      FUZNAME[18]:='STATG';
      FUZNAME[19]:='THR1E';
      FUZNAME[20]:='THR1F';
      FUZNAME[21]:='THR2E';
      FUZNAME[22]:='THR2F';
      FOR I:=1 TO NOFUZZYVARS DO
      BEGIN(*FOR*)
        FUZVARC[I]:=1;
        MAXLIMITC[I]:=10;
        MINLIMITC[I]:=0;
      END(*FOR*);
    END(*WITH*);
  END(*ELSE*);
  WITH FLIMIT DO
  BEGIN(*WITH*)
    REPEAT
      WRITELN;

```

```

WRITELN('THE CURRENT FUZZY VARIABLES FOLLOW: ');
WRITELN;
WRITELN(' FUZZY      INITIAL      MAXLIMIT      MINLIMIT');
WRITELN('VARIABLE    VALUE');
WRITELN('-----      -----      -----      -----');
WRITELN;
FOR I:=1 TO NOFUZZYVARS DO
BEGIN(*FOR*)
WRITELN(FUZNAME[I]:8,FUZVAR[I]:10:4,MAXLIMIT[I]:14:4,MINLIMIT[I]:14:4);
END(*FOR*);
WRITELN;
WRITELN;
WRITELN;
WRITELN('DO YOU WISH TO CHANGE ANY FUZZY VARIABLES? ');
WRITELN('ENTER "YES" TO CHANGE; ELSE ENTER "NO".');
WRITE('ANSWER = ');
READLN(ANSWER);
WRITELN;
IF ANSWER= 'YES' THEN
BEGIN(*IF*)
WRITELN('ENTER THE NAME OF THE VARIABLE TO BE CHANGED');
WRITE('VARIABLE NAME = ');
READLN(FNAME);
IF LENGTH(FNAME)<5 THEN
BEGIN(*IF*)
FOR I:=LENGTH(FNAME)TO 4 DO
BEGIN(*FOR*)
FNAME:=CONCAT(FNAME,' ');
END(*FOR*);
END(*IF*);
WRITELN;
J:=0;
FOR I:=1 TO NOFUZZYVARS DO
BEGIN(*FOR*)
IF FUZNAME[I]=FNAME THEN
BEGIN(*IF*)
J:=I;
END(*IF*);
END(*FOR*);
IF J=0 THEN
BEGIN(*IF*)
WRITELN('VARIABLE NAME DOES NOT EXIST. ');
END(*IF*);
ELSE
BEGIN(*ELSE*)
WRITELN('ENTER THE FOLLOWING VALUES FOR ',FNAME);
WRITELN;
WRITE('INITIAL VALUE = ');
READLN(FUZVAR[J]);
WRITE('MAXLIMIT = ');
READLN(MAXLIMIT[J]);
WRITE('MINLIMIT = ');
READLN(MINLIMIT[J]);
END(*ELSE*);

```

```

        WRITELN;
        WRITELN;
        END(*IF*);

UNTIL (ANSWER<>'YES');

REWRITE(OPTFUZZY);
FOR I:=1 TO NOFUZZYVARS DO
BEGIN(*FOR*)
    WRITELN(OPTFUZZY,FUZVARC[I],MAXLIMITC[I],MINLIMITC[I],FUZNAMEC[I]);
END(*FOR*);
CLOSE(OPTFUZZY);
END(*WITH*);

END(*IF*)
ELSE
BEGIN(*ELSE*)
    WRITELN;
    WRITELN ('INPUT EXACT SPELLING OF WORD WHOSE VARIABLES ');
    WRITELN('ARE TO BE CHANGED. ');
    WRITE ('WORD = ');
    READLN (WORD);
    FOR I:=(LENGTH(WORD)) TO (SPELLENGTH-1) DO
    BEGIN(*FOR*)
        WORD:=CONCAT(WORD,' ');
    END(*FOR*);
    WRITELN ;
    RESET (WFILE);
    TEMPWORD:=' ';
    WNUMB:=0;
    WHILE (NOT EOF(WFILE))AND(WORD<>TEMPWORD) DO
    BEGIN (*WHILE*)
        SEEK(WFILE,WNUMB);
        GET(WFILE);
        WDATA:=WFILE^;
        GET(WFILE);
        TEMPWORD:=WDATA.SPELLING;
        WNUMB:=WNUMB+ 1
    END (*WHILE*);
    WNUMB:=WNUMB-1;
    IF (WORD<>TEMPWORD) THEN
    BEGIN (*IF*)
        WRITELN;
        WRITELN ('WORD DOES NOT EXIST. ');
    END (*IF*)
    ELSE
    BEGIN (*ELSE*)
        RESET(FZOPT);
        SEEK(FZOPT,WNUMB);
        GET(FZOPT);
        FLIMIT:=FZOPT^;
        WITH FLIMIT DO
        BEGIN(*WITH*)

```

```

REPEAT
  WRITELN;
  WRITELN('THE CURRENT FUZZY VARIABLES FOLLOW: ');
  WRITELN;
  WRITELN('FUZZY      INITIAL      MAXLIMIT      MINLIMIT');
  WRITELN('VARIABLE      VALUE');
  WRITELN('-----      -----      -----      -----');
  WRITELN;
  FOR I:=1 TO NOFUZZYVARS DO
    BEGIN(*FOR*)
      WRITELN(FUZNAME[I]:8,FUZVAR[I]:10:4,MAXLIMIT[I]:14:4,MINLIMIT[I]:14:4);
    END(*FOR*);
  WRITELN;
  WRITELN;
  WRITELN;
  WRITELN('DO YOU WISH TO CHANGE ANY FUZZY VARIABLES? ');
  WRITELN('ENTER "YES" TO CHANGE; ELSE ENTER "NO".');
  WRITE('ANSWER = ');
  READLN(ANSWER);
  WRITELN;
  IF ANSWER= 'YES' THEN
    BEGIN(*IF*)
      WRITELN('ENTER THE NAME OF THE VARIABLE TO BE CHANGED');
      WRITE('VARIABLE NAME = ');
      READLN(FNAME);
      IF LENGTH(FNAME)<5 THEN
        BEGIN(*IF*)
          FOR I:=LENGTH(FNAME)TO 4 DO
            BEGIN(*FOR*)
              FNAME:=CONCAT(FNAME,' ');
            END(*FOR*);
          END(*IF*);
          WRITELN;
          J:=0;
          FOR I:=1 TO NOFUZZYVARS DO
            BEGIN(*FOR*)
              IF FUZNAME[I]=FNAME THEN
                BEGIN(*IF*)
                  J:=1;
                END(*IF*);
            END(*FOR*);
          IF J=0 THEN
            BEGIN(*IF*)
              WRITELN('VARIABLE NAME DOES NOT EXIST. ');
            END(*IF*);
          ELSE
            BEGIN(*ELSE*)
              WRITELN('ENTER THE FOLLOWING VALUES FOR ',FNAME);
              WRITELN;
              WRITE('INITIAL VALUE = ');
              READLN(FUZVAR[J]);
              WRITE('MAXLIMIT = ');
              READLN(MAXLIMIT[J]);
              WRITE('MINLIMIT = ');
            END(*ELSE*);
          END(*IF*);
        END(*IF*);
      END(*IF*);
    END(*IF*);
  END(*IF*);

```

```

        READLN(MINLIMIT[J]);
        END(*ELSE*);
        WRITELN;
        WRITELN;
        END(*IF*);

    UNTIL (ANSWER<>'YES');

    SEEK(FZOPT,WNUMB);
    FZOPT^:=FLIMIT;
    PUT(FZOPT);
    END(*WITH*);
    END(*ELSE*);
    END(*ELSE*);

END(*PROCEDURE CHNGEOPT*);

```

(*****)

PROCEDURE AUTODET DESCRIPTION

THIS PROCEDURE DETERMINES A PHONEME REPRESENTATION FOR A GIVEN WORD BASED ON THE TRAINING SPEECH FILES STORED IN FILE SPECH. THIS PROCEDURE ESSENTIALLY PROVIDES THE FOLLOWING FOUR OPERATIONS:

OPERATION	DESCRIPTION
-----	-----
AUTODET	- DETERMINES AN INITIAL PHONEME REPRESENTATION BASED ON THE POSITION AND NUMBER OF TIMES A PHONEME OCCURS IN THE TRAINING FILES.
CORRECTR	- USES THE STATISTICS COLLECTED FROM THE PREVIOUS REPRESENTATION TO IMPROVE IT BY INSERTING THE PHONEMES WHOSE INSERTION AND SUBSTITUTION STATISTICS WERE HIGHER THAN THOSE OF THE INITIAL REPRESENTATION.
PICKBEST	- USES THE STATISTICS COLLECTED FROM THE PREVIOUS REPRESENTATION TO IMPROVE IT BY REPLACING THE PHONEMES WHOSE INSERTION AND SUBSTITUTION STATISTICS ARE LOWER THAN ANOTHER PHONEME'S.
ADJUSTDEL	- ALTERS THE CURRENT PHONEME REPRESENTATION BASED ON THE NUMBER OF TIMES EACH WORD PHONEME WAS DELETED. THE DELETION STATISTICS WERE COLLECTED USING THE PREVIOUS PHONEME REPRESENTATION.

PROCEDURE CONSTANTS

CONSTANT	DESCRIPTION
-----	-----
THRESHHOLD	- CONSTANT USED IN PROCEDURE FINDEND TO DETERMINE WHEN A WORD PHONEME ENDS.

MAIN GLOBAL PROCEDURE VARIABLES

VARIABLE	DESCRIPTION
-----	-----
PHINT	- INTEGER INDICATING THE CURRENT WORD PHONEME.
DPHREP	- ARRAY USED TO STORE THE WORD PHONEME REPRESENTATION.
DNOSPOKE	- NUMBER OF SPEECH FILES IN THE WORD'S TRAINING SET.
PHSCORE	- ARRAY (CONSISTING OF SCORES FOR EACH PROTOTYPE PHONEME)

PHCNT USED TO DETERMINE THE NEXT WORD PHONEME.
 - THE NUMBER OF TIMES A PHONEME IS CONSECUTIVELY CHOSEN
 AS THE NEXT WORD PHONEME.

*****}

SEGMENT PROCEDURE AUTODET(VAR NOWORD:INTEGER; NOPHVECTORS:INTEGER;
 VAR CHNGEFLAG:BOOLEAN);

CONST
 THRESHHOLD=0.30;

TYPE
 PHARRAY=ARRAY[1..NOPHONES] OF REAL;

VAR
 PHINT:INTEGER;
 DPHREP:ARRAY[1..MAXPHONES] OF INTEGER;
 DNOSPOKE:INTEGER;
 PHSCORE:PHARRAY;
 PHCNT:ARRAY[1..MAXPHONES] OF INTEGER;
 WORDDET,WTEMP:STRING[SPELLENGTH];
 J,K,I,INT,JINT:INTEGER;
 DETRECORD,DETREPCNT:INTEGER;
 COUNT:INTEGER;
 TEMP:INTEGER;
 FLAG:BOOLEAN;
 INS:ARRAY[0..NOPHONES] OF REAL;
 NJEOPS:INTEGER;
 FINISHED:BOOLEAN;

{*****}

PROCEDURE FIXREP DESCRIPTION

 SINCE THE WORD RECOGNITION ALGORITHM EXAMINES 3 WORD PHONEMES
 AT A TIME, TO DETERMINE TRANSITIONS BETWEEN PHONEMES, IT IS NECESSARY

(OR EXTREMELY DESIRABLE) THAT ANY COMBINATION OF THREE ADJACENT PHONEMES IN THE WORD REPRESENTATION BE UNIQUE. THIS PROCEDURE INSURES THIS UNIQUENESS BY ELIMINATING ALL PHONEMES THAT ARE A DUPLICATE OF ONE OF THE PRECEDING TWO WORD PHONEMES.

*****)

PROCEDURE FIXREP;

VAR

I,J:INTEGER;
CFLAG:BOOLEAN;

BEGIN(*FIXREP*)

CFLAG:=FALSE;
WITH WDATA DO
BEGIN(*WITH*)
IF PHINT>2 THEN
BEGIN(*IF*)

{Determine if any two consecutive word phonemes are duplicated, and eliminate the second duplicate.}

FOR I:=3 TO PHINT DO
BEGIN(*FOR*)
IF PHREP[I-2]=PHREP[I] THEN
BEGIN(*IF*)
FOR J:=1 TO (PHINT-1) DO
BEGIN(*FOR*)
PHREP[J]:=PHREP[J+1];
END(*FOR*);
PHREP[PHINT]:=0;
PHINT:=PHINT-1;
CFLAG:=TRUE;
END(*IF*);
END(*FOR*);

{Determine if the first and third word phonemes of any three consecutive phonemes are duplicates, and eliminate the second duplicate.}

FOR I:=2 TO PHINT DO
BEGIN(*FOR*)
IF PHREP[I-1]=PHREP[I] THEN
BEGIN(*IF*)
FOR J:=1 TO (PHINT) DO
BEGIN(*FOR*)

```

        PHREP[J-1]:=PHREP[J];
      END(*FOR*);
      PHREP[PHINT]:=0;
      PHINT:=PHINT-1;
      CFLAG:=TRUE;
    END(*IF*);
  END(*FOR*);
END(*IF*);
END(*WITH*);
IF CFLAG=TRUE THEN
BEGIN(*IF*)

  {Reevaluate representation (if any changes were made)
  to eliminate any duplicates that could have been
  created.}

  FIXREP;
END(*IF*);

END(*PROCEDURE FIXREP*);

```

{*****}

PROCEDURE MAXCHOICE DESCRIPTION

THIS PROCEDURE ASSIGNS THE NEXT WORD PHONEME FOR THE
INITIAL REPRESENTATION BY CHOOSING THE PHONEME WITH THE HIGHEST
SCORE IN THE ARRAY PHSCORE.

*****}

PROCEDURE MAXCHOICE;

VAR

MAXCH:REAL;
J:INTEGER;

BEGIN(*MAXCHOICE*)

{Determine phoneme with highest score.}

```

MAXCH:=0;
FOR J:= 1 TO NOPHONEMES DO
BEGIN(*FOR*)
  IF PHSCORE[J]>MAXCH THEN
  BEGIN(*IF*)
    WDATA.PHREP[PHINT]:=J;
    MAXCH:= PHSCORE[J];
  END(*IF*);
  PHSCORE[J]:=0;
END(*FOR*);
WRITELN('PHONEME CHOSEN= ',WDATA.PHREP[PHINT]:2,' PHINT= ',PHINT:2,
        ' SCORE= ',MAXCH);

(Update word phoneme representation.)

IF PHINT(>)1 THEN
BEGIN(*IF*)
  IF WDATA.PHREP[PHINT]=WDATA.PHREP[PHINT-1] THEN
  BEGIN(*IF*)
    WDATA.PHREP[PHINT]:=0;
    PHINT:=PHINT-1;
    PHCNT[PHINT]:=PHCNT[PHINT]+1;
  END(*IF*);
END(*IF*);
PHINT:=PHINT+1;

END(*PROCEDURE MAXCHOICE*);

```

{*****}

PROCEDURE SUMXVECTORS DESCRIPTION

THIS PROCEDURE SUMS THE SCORES OF EACH PHONEME OCCURRING IN THE NEXT "NOPHVECTORS" VECTORS OF THE CURRENT SPEECH FILE. IT SHOULD BE NOTED THAT THIS SUM IS ALSO SUMMED WITH THE PREVIOUS SUMS OBTAINED FOR THE PREVIOUS SPEECH FILES. THIS IS DONE TO DETERMINE A WEIGHT FOR EACH PHONEME THAT IS USED BY PROCEDURE MAXCHOICE TO CHOOSE THE NEXT WORD PHONEME.

*****}

```

PROCEDURE SUMXVECTORS(VAR PHSCORE:PHARRAY);
VAR
    I,J:INTEGER;
    VSCORE,ARGUMENT:REAL;

BEGIN(*SUMXVECTORS*)

    DETREPCNT:=1;

    {Initialize array used to keep track of
    phoneme scores.}

    FOR I:=1 TO NOPHONEMES DO
    BEGIN(*FOR*)
        INSC[I]:=0.0;
    END(*FOR*);
    FLAG:=TRUE;
    WITH IOUT DO
    BEGIN(*WITH*)
        WHILE (FLAG=TRUE) DO
        BEGIN(*WHILE*)

            {Update score for each phoneme occurring in
            next "NOPHVECTORS" vectors.}

            FOR I:=1 TO NOGUESSES DO
            BEGIN(*FOR*)
                WITH IOUT DO
                BEGIN(*WITH*)
                    TEMP:=ICHPHONE[I,K];
                    ARGUMENT:=ISFACTOR[K]/SFE;
                    VSCORE:=(1-ICHVALUE[I,K]/100)/CHVE;
                    VSCORE:=(1-POWER(VSCORE,CHVF))*(1-POWER(ARGUMENT,SFF));
                    INSC[TEMP]:=VSCORE+INSC[TEMP];
                END(*WITH*);
            END(*FOR*);
            K:=K+1;
            DETREPCNT:=DETREPCNT+1;
            IF ((K)RCRDLENGTH) AND (CONTFLAG=1) THEN
            BEGIN(*IF*)
                GET(SPECH);
                IOUT:=SPECH^;
                K:=1;
            END(*IF*)
            ELSE IF (K)RCRDLENGTH)AND(CONTFLAG=0)THEN
            BEGIN(*ELSE*)
                K:=1;
                FLAG:=FALSE;
            END(*ELSE*);
            IF (DETREPCNT>NOPHVECTORS) THEN
            BEGIN(*IF*)
                DETREPCNT:=1;
                FLAG:=FALSE;

```

{Update variable that keeps track of each phoneme's score for each of the word's training files for the word phoneme being determined.}

```

FOR J:=1 TO NOPHONEMES DO
  BEGIN(*FOR*)
    IF INSLJ>0.0 THEN
      BEGIN(*IF*)
        PHSCORE[J]:= PHSCORE[J]+
                                INSLJ;
      END(*IF*);
    END(*FOR*);
  END(*IF*);
END(*WHILE*);
IF DETREPCNT>1 THEN
  BEGIN(*IF*)
    FOR J:=1 TO NOPHONEMES DO
      BEGIN(*FOR*)
        IF INSLJ>0.0 THEN
          BEGIN(*IF*)
            PHSCORE[J]:=PHSCORE[J]+
                                INSLJ;
          END(*IF*);
        END(*FOR*);
      END(*IF*);
    END(*WITH*);
  END(*PROCEDURE SUMXVECTORS*);

```

{*****}

PROCEDURE FINDEND DESCRIPTION

THIS PROCEDURE ATTEMPTS TO FIND THE STARTING POINT OF THE NEXT WORD PHONEME IN EACH OF THE WORD'S TRAINING FILES BY USING AN ALGORITHM THAT IS SIMILAR TO THE ACTUAL RECOGNITION ALGORITHM. THE MAIN DIFFERENCE BETWEEN THE ALGORITHM USED IN THIS PROCEDURE, IS THAT A THRESHOLD MUST BE USED IN DETERMINING WHEN A WORD PHONEME ENDS, SINCE THE NEXT WORD PHONEME IS UNKNOWN. WHEN THE STARTING VECTOR OF THE NEXT PHONEME IN A SPEECH FILE IS DETERMINED, PROCEDURE SUMXVECTORS IS CALLED. AFTER ALL THE SPEECH FILES HAVE UNDERGONE THIS PROCESS, PROCEDURE MAXCHOICE IS CALLED. THIS PROCESS IS REPEATED UNTIL THE WORD PHONEME REPRESENTATION ACCOUNTS FOR EVERY VECTOR IN EVERY TRAINING FILE. IT SHOULD BE EMPHASIZED THAT THIS PROCEDURE IS USED ONLY TO ARRIVE AT AN

INITIAL WORD REPRESENTATION.

*****}

PROCEDURE FINDEND;

VAR

PRPHONE:REAL;
I,J:INTEGER;
ENDOFFILE:BOOLEAN;
VSCORE,ARGUMENT:REAL;
PHINS:ARRAY[0..NOPHONEMES] OF REAL;
MAX:REAL;
PHTMP:INTEGER;

BEGIN(*FINDEND*)

RESET(SPECH);
NOEofs:=0;
DETRECORD:=0;
WHILE NOT EOF(SPECH) DO
BEGIN(*WHILE*)
SEEK(SPECH,DETRECORD);
GET(SPECH);
IOUT:=SPECH^;
IF IOUT.SPELL=WORDDET THEN
BEGIN(*IF*)
FOR J:=1 TO NOPHONEMES DO
BEGIN(*FOR*)
PHINS[J]:=0.0;
END(*FOR*);
BEGINREC:=DETRECORD;
INT:=0;
K:=1;
REPEAT
FINISHED:=FALSE;
ENDOFFILE:=TRUE;
INT:=INT+1;
K:=(PHCNT[INT]*3)+K;
IF (K)RCRDLENGTH) AND (IOUT.CONTFILAG=1) THEN
BEGIN(*IF*)
GET(SPECH);
IOUT:=SPECH^;
DETRECORD:=DETRECORD+1;
K:=K-RCRDLENGTH;
END(*IF*)
ELSE IF (K)RCRDLENGTH) AND (IOUT.CONTFILAG=0) THEN
BEGIN(*ELSE*)
K:=40;

END(*ELSE*);

{Determine where the last word phoneme ends in
the speech file.}

REPEAT

PRPHONE:=0;

FOR J:= 1 TO NOGUESSES DO

BEGIN(*FOR*)

IF (WDATA.PHREP[INT]=IOUT.ICHPHONE[J,K]) THEN

BEGIN(*IF*)

WITH IOUT DO

BEGIN(*WITH*)

PHTEMP:=ICHPHONE[J,K];

ARGUMENT:=ISFACTOR[K]/SFE;

VSCORE:=(1-ICHVALUE[J,K]/100)/CHVE;

VSCORE:=(1-POWER(VSCORE,CHVF))*(1-
POWER(ARGUMENT,SFF));

PRPHONE:=VSCORE;

END(*WITH*);

END(*IF*);

END(*FOR*);

{If the current word phoneme did not occur
in the choices for the current vector, then test
to determine if it occurs in the next NOPHVECTORS
vectors.}

IF PRPHONE=0 THEN

BEGIN(*IF*)

COUNT:=0;

FOR I:=1 TO NOPHVECTORS DO

BEGIN(*FOR*)

K:=K+1;

IF (K)RCRDLENGTH)AND(IOUT.CONTFLAG=1) THEN

BEGIN(*IF*)

GET(SPECH);

IOUT:=SPECH^;

DETRECORD:=DETRECORD+1;

K:=1;

END(*IF*)

ELSE IF (K)RCRDLENGTH)AND(IOUT.CONTFLAG=0) THEN

BEGIN(*ELSE*)

IF COUNT=0 THEN

COUNT:=1;

K:=41;

FINISHED:=TRUE;

END(*ELSE*);

IF FINISHED=FALSE THEN

BEGIN(*IF*)

COUNT:=COUNT+1;

FOR J:=1 TO NOGUESSES DO

BEGIN(*FOR*)

WITH IOUT DO

```

        BEGIN(*WITH*)
            PHTEMP:=ICHPHONE[J,K];
            ARGUMENT:=ISFACTOR[K]/SFE;
            VSCORE:=(1-ICHVALUE[J,K]/100)/CHVE;
            VSCORE:=(1-POWER(VSCORE,CHVF))*(1-
                POWER(ARGUMENT,SFF));
            PHINS[PHTEMP]:=VSCORE+PHINS[PHTEMP];
        END(*WITH*);
    END(*FOR*);
    END(*IF*);
    END(*FOR*);
    IF (PHINS[DATA.PHREP[INT]]/COUNT)<THRESHOLD THEN
    BEGIN(*IF*)
        FINISHED:=TRUE;
    END(*IF*);
    MAX:=0;
    FOR J:= 1 TO NOPHONEMES DO
    BEGIN(*FOR*)
        IF PHINS[J]>MAX THEN
        BEGIN(*IF*)
            PHTEMP:=J;
            MAX:= PHINS[J];
        END(*IF*);
        PHINS[J]:=0;
    END(*FOR*);
    IF (K-COUNT < 0) THEN
    BEGIN(*IF*)
        SEEK(SPECH,(DETRECORD-1));
        GET(SPECH);
        IOUT:=SPECH^;
        K:=K-COUNT+RCRDLENGTH;
    END(*IF*)
    ELSE
    BEGIN(*ELSE*)
        K:=K-COUNT;
    END(*ELSE*);
    END(*IF*);
    K:=K+1;
    IF (K)>RCRDLENGTH)AND (IOUT.CONTFLAG=1) THEN
    BEGIN(*IF*)
        GET(SPECH);
        IOUT:=SPECH^;
        K:=1;
    END(*IF*)
    ELSE IF (K)>RCRDLENGTH)AND(IOUT.CONTFLAG=0)THEN
    BEGIN(*ELSE*)
        ENDOFFILE:=FALSE;
        FINISHED:=TRUE;
    END(*ELSE*);
    UNTIL (FINISHED=TRUE);
    UNTIL (INT=(PHINT-1));
    IF ENDOFFILE=TRUE THEN
    BEGIN(*IF*)
        SUMXVECTORS(PHSCORE);

```

```

END(*IF*)
ELSE
BEGIN(*ELSE*)
    NOEOFS:=NOEOFS+1;
END(*ELSE*);
DETRCORD:=BEGINREC;
SEEK(SPECH,DETRCORD);
REPEAT
    GET(SPECH);
    IOUT:=SPECH^;
    DETRECORD:=DETRCORD+1;
UNTIL (IOUT,CONTFLAG=0);
GET(SPECH);
END(*IF*)
ELSE
BEGIN(*ELSE*)
    DETRECORD:=DETRCORD+1;
    GET(SPECH);
END(*ELSE*);
END(*WHILE*);
WRITELN('NO. EOFS= ',NOEOFS);
IF NOEOFS<DNOSPOKE THEN
BEGIN(*IF*)
    MAXCHOICE;
    IF PHINT<=MAXPHONES THEN
    BEGIN(*IF*)
        FINDEND;
    END(*IF*)
    ELSE
    BEGIN(*ELSE*)
        PHINT:=PHINT-1;
    END(*ELSE*);
END(*IF*);
END(*PROCEDURE FINDEND*);

```

<*****

PROCEDURE CORRECTREP DESCRIPTION

THIS PROCEDURE ADDS PHONEMES TO THE CURRENT WORD REPRESENTATION ON THE BASIS OF THE STATISTICS COLLECTED USING THE PREVIOUS WORD REPRESENTATION. THE PHONEMES ARE INSERTED BASED ON THE SUM OF THE

INSERTION AND SUBSTITUTION SCORES FOR EACH WORD PHONEME.

*****}

PROCEDURE CORRECTREP(NOWORD:INTEGER; VAR CHNGEFLAG:BOOLEAN);

VAR

CFACOR,IMAX,I,J:INTEGER;
SUBINS:ARRAY[0..MAXPHPLUS1,1..NOPHONEMES] OF REAL;
MAX:REAL;
DPHREP:ARRAY[1..MAXPHONES] OF INTEGER;

BEGIN(*CORRECTREP*)

WRITELN('CORRECTREP');
SEEK(SFILE,NOWORD);
GET(SFILE);
WSTATS:=SFILE^;
SEEK(WFILE,NOWORD);
GET(WFILE);
WDATA:=WFILE^;

{Save initial word representation.}

FOR I:=1 TO MAXPHONES DO
BEGIN(*FOR*)
DPHREP[I]:=WDATA.PHREP[I];
END(*FOR*);

{Initialize the variables used to indicate
each phoneme's substitution and insertion
statistics for each word phoneme.}

FOR I:=1 TO NOPHONEMES DO
BEGIN(*FOR*)
SUBINS[0,I]:=0.0;
SUBINS[(MAXPHONES+1),I]:=0.0;
END(*FOR*);
FOR J:=1 TO MAXPHONES DO
BEGIN(*FOR*)
FOR I:=1 TO NOPHONEMES DO
BEGIN(*FOR*)
SUBINS[J,I]:=WSTATS.XSUBY[J,I]+WSTATS.XINAFTZ[J,I];
END(*FOR*);
END(*FOR*);
PHINT:=1;
CFACOR:=0;

```

WITH WDATA DO
BEGIN(*WITH*)
  WHILE (PHREP[PHINT](>0)AND(PHINT<=MAXPHONES)DO
  BEGIN(*WHILE*)

    {Test to see if statistics indicate that the
    current word phoneme is the best.}

    MAX:=0;
    IMAX:=0;
    FOR I:=1 TO NOPHONEMES DO
    BEGIN(*FOR*)
      IF (SUBINS[PHINT,I]>MAX)OR((IMAX=PHREP[PHINT])AND
      (SUBINS[PHINT,I]=MAX)) THEN
      BEGIN(*IF*)
        MAX:=SUBINS[PHINT,I];
        IMAX:=I;
      END(*IF*);
    END(*FOR*);

    {If the current word phoneme is not the best,
    add another phoneme to the representation.}

    IF (IMAX<>PHREP[PHINT+CFACTOR])AND(IMAX<>0) THEN
    BEGIN(*IF*)
      CHNGEFLAG:=TRUE;
      IF PHREP[MAXPHONES](>0) THEN
      BEGIN(*IF*)
        FOR J:=1 TO MAXPHONES DO
        BEGIN(*FOR*)
          DPHREP[J]:=PHREP[J];
        END(*FOR*);

        {Determine if the new phoneme should be added before
        or after the current word phoneme based on the
        statistics for the adjacent word phonemes; and make
        the appropriate revision.}

        IF (SUBINS[(PHINT-1),IMAX]=SUBINS[(PHINT+1),IMAX])AND
        (SUBINS[(PHINT-1),IMAX]>SUBINS[(PHINT-1),
        (PHREP[PHINT+CFACTOR])]) THEN
        BEGIN(*IF*)
          FOR J:=(PHINT+CFACTOR) TO (MAXPHONES-1) DO
          BEGIN(*FOR*)
            PHREP[J+1]:=DPHREP[J];
          END(*FOR*);
          PHREP[PHINT+CFACTOR]:=IMAX;
        END(*IF*)
      ELSE
      BEGIN(*ELSE*)
        FOR J:=(CFACTOR+PHINT+1) TO (MAXPHONES-1) DO
        BEGIN(*FOR*)
          PHREP[J+1]:=DPHREP[J];
        END(*FOR*);
      END(*ELSE*);
    END(*IF*);
  END(*WHILE*);
END(*WITH*);

```

```

        PHREP[PHINT+1+CFACTOR]:=IMAX;
    END(*ELSE*);
    CFACTOR:=CFACTOR+1;
END(*IF*)
ELSE
BEGIN(*ELSE*)

    {If the maximum number of word phonemes has been
    reached, simply replace the current word phoneme
    with the one indicated by the statistics.}

    PHREP[PHINT+CFACTOR]:=IMAX;
    END(*ELSE*);
END(*IF*);

    PHINT:=PHINT+1;
    END(*WHILE*);
END(*WITH*);

FIXREP;
SEEK(WFILE,NOWORD);
WFILE^:=WDATA;
PUT(WFILE);

END(*PROCEDURE CORRECTREP*);

```

{*****}

PROCEDURE PICKBEST DESCRIPTION

THIS PROCEDURE USES THE SUM OF EACH WORD PHONEME'S INSERTION
AND SUBSTITUTION STATISTICS, TO REPLACE ALL WORD PHONEMES WITH
THE PHONEME WHOSE SUMMATION IS HIGHEST.

*****}

```
PROCEDURE PICKBEST(NOWORD:INTEGER; VAR CHNGEFLAG:BOOLEAN);
```

```
VAR
```

```
    IMAX,I,J:INTEGER;
```

```
    MAX:REAL;
```

```
    SUBINS:ARRAY[0..MAXPHPLUS1,1..NOPHONEMES] OF REAL;
```

```
BEGIN(*PICKBEST*)
```

```
    WRITELN('PICKBEST');
```

```
    SEEK(SFILE,NOWORD);
```

```
    GET(SFILE);
```

```
    WSTATS:=SFILE^;
```

```
    SEEK(WFILE,NOWORD);
```

```
    GET(WFILE);
```

```
    WDATA:=WFILE^;
```

```
    {Initialize variable array that indicates  
    each phonemes substitution and insertion  
    plausibilities for each word phoneme.}
```

```
    FOR I:=1 TO NOPHONEMES DO
```

```
    BEGIN(*FOR*)
```

```
        SUBINS[0,I]:=0.0;
```

```
        SUBINS[MAXPHPLUS1,I]:=0.0;
```

```
    END(*FOR*);
```

```
    FOR J:=1 TO MAXPHONES DO
```

```
    BEGIN(*FOR*)
```

```
        FOR I:=1 TO NOPHONEMES DO
```

```
        BEGIN(*FOR*)
```

```
            SUBINS[J,I]:=WSTATS.XSUBY[J,I]+WSTATS.XINFTZ[J,I];
```

```
        END(*FOR*);
```

```
    END(*FOR*);
```

```
    PHINT:=1;
```

```
    WITH WDATA DO
```

```
    BEGIN(*WITH*)
```

```
        {Test to see if the statistics indicate  
        that the current word phoneme is the best.}
```

```
    WHILE (PHREP[PHINT]<>0)AND(PHINT<=MAXPHONES)DO
```

```
    BEGIN(*WHILE*)
```

```
        MAX:=0;
```

```
        IMAX:=0;
```

```
        FOR I:=1 TO NOPHONEMES DO
```

```
        BEGIN(*FOR*)
```

```
            IF (SUBINS[PHINT,I]>MAX)OR((IMAX=PHREP[PHINT])AND  
            (SUBINS[PHINT,I]=MAX)) THEN
```

```
            BEGIN(*IF*)
```

```
                MAX:=SUBINS[PHINT,I];
```

```
                IMAX:=I;
```

```
            END(*IF*);
```

```

END(*FOR*);

(If current word phoneme is not the best,
then replace it with the one the statistics
suggest is better.)

IF (IMAX(>PHREP[PHINT])AND(IMAX(>0) THEN
BEGIN(*IF*)
  CHNGEFLAG:=TRUE;
  PHREP[PHINT]:=IMAX;
END(*IF*);
  PHINT:=PHINT+1;
END(*WHILE*);
END(*WITH*);

FIXREP;
SEEK(WFILE,NOWORD);
WFILE^:=WDATA;
PUT(WFILE);

```

```

END(*PROCEDURE PICKBEST*);

```

```

(*****

```

``` PROCEDURE ADJUSTDEL DESCRIPTION ----- ```

THIS PROCEDURE DELETES WORD PHONEMES BASED ON THE NUMBER
 OF TIMES EACH PHONEME WAS DELETED. THE DELETION STATISTICS WERE
 COLLECTED USING THE CURRENT REPRESENTATION.

```

*****

```

```

PROCEDURE ADJUSTDEL(NOWORD:INTEGER; VAR CHNGEFLAG:BOOLEAN);

```

```

VAR
  CFACTOR,I,J:INTEGER;

```

BEGIN(*ADJUSTDEL*)

```
WRITELN('ADJUSTDEL');  
SEEK(SFILE,NOWORD);  
GET(SFILE);  
WSTATS:=SFILE^;  
SEEK(WFILE,NOWORD);  
GET(WFILE);  
WDATA:=WFILE^;
```

{Save current word representation.}

```
FOR I:=1 TO MAXPHONES DO  
  BEGIN(*FOR*)  
    DPHREP[I]:=WDATA.PHREP[I];  
  END(*FOR*);  
CFACOR:=0;  
FOR I:=1 TO MAXPHONES DO  
  BEGIN(*FOR*)
```

{Delete all current word phonemes that meet
the following test.}

```
  IF WSTATS.NDEL[I]>ROUND(WSTATS.NOSPOKE/2) THEN  
    BEGIN(*IF*)  
      CHNGEFLAG:=TRUE;  
      FOR J:=(I-CFACOR) TO (MAXPHONES-1) DO  
        BEGIN(*FOR*)  
          DPHREP[J]:=DPHREP[J+1];  
        END(*FOR*);  
        DPHREP[MAXPHONES]:=0;  
        CFACOR:=CFACOR+1;  
      END(*IF*);  
    END(*FOR*);  
  FOR I:=1 TO MAXPHONES DO  
    BEGIN(*FOR*)  
      WDATA.PHREP[I]:=DPHREP[I];  
    END(*FOR*);  
  PHINT:=0;  
  REPEAT  
    PHINT:=PHINT+1;  
  UNTIL (WDATA.PHREP[PHINT]=0)OR(PHINT=MAXPHONES);  
  IF WDATA.PHREP[PHINT]=0 THEN  
    PHINT:=PHINT-1;  
  FIXREP;  
  SEEK(WFILE,NOWORD);  
  WFILE^:=WDATA;  
  PUT(WFILE);
```

END(*PROCEDURE ADJUSTDEL*);

BEGIN(*AUTODET*)

IF OPERATION='AUTODET' THEN

BEGIN(*IF*)

WRITELN('AUTODET');

SEEK(WFILE,NOWORD);

GET(WFILE);

WDATA:=WFILE^;

WORDDET:=WDATA.SPELLING;

{Initialize the word representation to zero's.}

FOR I:=1 TO MAXPHONES DO

BEGIN(*FOR*)

WDATA.PHREP[I]:=0;

PHCNT[I]:=1;

END(*FOR*);

DNOSPOKE:=0;

FOR J:= 1 TO NOPHONEMES DO

BEGIN (*FOR*)

PHSCORE[J]:=0.0;

END (*FOR*);

RESET(SPECH);

DETRERCORD:=0;

{Determine number of training files stored for the word, and determine the first word phoneme in the word representation.}

WHILE NOT EOF(SPECH) DO

BEGIN(*WHILE*)

SEEK(SPECH,DETRERCORD);

GET(SPECH);

IOUT:=SPECH^;

IF IOUT.SPELL=WORDDET THEN

BEGIN(*IF*)

DNOSPOKE:= DNOSPOKE+1;

WRITELN('NUMBER OF SPEECHFILES FOR WORD = ', DNOSPOKE);

BEGINREC:=DETRERCORD;

PHINT:=1;

K:=1;

SUMXVECTORS(PHSCORE);

DETRERCORD:=BEGINREC;

SEEK(SPECH,DETRERCORD);

REPEAT

GET(SPECH);

IOUT:=SPECH^;

DETRERCORD:=DETRERCORD+1;

UNTIL (IOUT.CONTFLAG=0);

GET(SPECH);

END(*IF*)

ELSE

BEGIN(*ELSE*)

DETRERCORD:=DETRERCORD+1;

```

        GET(SPECH);
        END(*ELSE*);
    END (*WHILE*);

    WRITELN;
    WRITELN;

    MAXCHOICE;
    FINDEND;

    FIXREP;

    SEEK(WFILE,NOWORD);
    WFILE^:=WDATA;
    PUT(WFILE);

END(*IF*)
ELSE IF OPERATION='CORRECTR' THEN
BEGIN(*ELSE*)
    CORRECTREP(NOWORD,CHNGEFLAG);
END(*ELSE*)
ELSE IF OPERATION='PICKBEST' THEN
BEGIN(*ELSE*)
    PICKBEST(NOWORD,CHNGEFLAG);
END(*ELSE*)
ELSE IF OPERATION='ADJUSTDE' THEN
BEGIN(*ELSE*)
    ADJUSTDEL(NOWORD,CHNGEFLAG);
END(*ELSE*)
ELSE
BEGIN(*ELSE*)
    WRITELN('ERROR; NO OPERATION SPECIFIED.');
```

9

```

END(*ELSE*);

FOR I:=1 TO MAXPHONES DO
BEGIN(*FOR*)
    WRITE(STATSOUT,WDATA.PHREP[I]:3);
END(*FOR*);
WRITELN(STATSOUT,WDATA.SPELLING);

END (*PROCEDURE AUTODET*);
```

{*****}

PROCEDURE STOREWORD DESCRIPTION

THIS PROCEDURE ALLOWS A USER TO STORE A NUMBER OF SPEECH FILES IN FILE SPECH OR FILE RECOG.

*****}

SEGMENT PROCEDURE STORESPEECH;

VAR

NOWORD,I:INTEGER;
ANS,ANSWER:STRING[3];
TEMPWORD:STRING[SPELENGTH];
FLAG:BOOLEAN;

BEGIN(*STORESPEECH*)

INITFILE;
ENDFLAG:=FALSE;
WRITELN;
WRITELN('ENTER "YES" TO STORE SPEECH FILES FOR');
WRITELN('INITIALIZING WORD STATISTICS; ELSE ENTER ');
WRITELN('"NO" TO STORE SPEECH FILES FOR FUTURE');
WRITELN('RECOGNITION SCORING.');

WRITE('ANSWER = ');
READLN(ANSWER);
WRITELN;
IF ANSWER='YES' THEN
BEGIN(*IF*)
REPEAT
WINIT1;
IF ENDFLAG=FALSE THEN
BEGIN(*IF*)
STOREWORD;
END(*IF*);
WRITELN;
WRITELN('ENTER "YES" IF YOU WOULD LIKE TO ADD');
WRITELN('ANOTHER SPEECH FILE; ELSE ENTER "NO".');

WRITE('ANSWER = ');
READLN(ANS);
WRITELN;
WRITELN;
UNTIL ANS<>'YES';

```

END(*IF*)
ELSE
BEGIN(*ELSE*)
    FILENAME:=CONCAT(SPEAKER,'RECOG');
    FILETITLE(SPECH,FILENAME);

    REPEAT
        RESET(SPECH);
        WRITELN;
        WRITELN('INPUT NAME OF SPEECHFILE');
        WRITE('FILENAME = ');
        READLN(FILENAME);
        WRITELN;
        FILETITLE(OUT2,FILENAME);
        RESET (OUT2);
        WITH IOUT DO
            BEGIN (*WITH*)

                {Determine if word given in speech file
                 is in the current vocabulary. If it is
                 not then allow user to change the word's
                 spelling.}

                READLN(OUT2,SPELL);
                FOR I:= (LENGTH(SPELL)) TO (SPELLENGTH-1) DO
                    BEGIN (*FOR*)
                        SPELL:=CONCAT(SPELL,' ');
                    END (*FOR*);
                REPEAT
                    RESET (WFILE);
                    FLAG:=FALSE;
                    TEMPWORD:=' ';
                    NOWORD:=0;
                    WHILE (NOT EOF(WFILE))AND(SPELL<>TEMPWORD) DO
                        BEGIN (*WHILE*)
                            SEEK(WFILE,NOWORD);
                            GET(WFILE);
                            WDATA:=WFILE^;
                            GET(WFILE);
                            TEMPWORD:=WDATA.SPELLING;
                            NOWORD:=NOWORD+ 1
                        END (*WHILE*);
                    WNUMB:=NOWORD-1;
                    IF (SPELL<>TEMPWORD) THEN
                        BEGIN (*IF*)
                            WRITELN;
                            WRITELN ('NONE OF THE WORDS IN THE VOCABULARY');
                            WRITELN ('AGREE WITH THE SPELLING GIVEN IN THE');
                            WRITE ('FILE "',FILENAME,'" THE SPELLING IN ',FILENAME,' IS');
                            WRITELN ('"',SPELL,'" ');
                            WRITELN ('DO YOU WANT TO CHANGE THE "',FILENAME,'" SPELLING?');
                            WRITELN ('ENTER "YES" TO CHANGE, ELSE ENTER "NO"');
                            WRITE ('ANSWER = ');
                            READLN(ANSWER);
                        END (*IF*);
                    END (*WHILE*);
                END (*FOR*);
            END (*WITH*);
        END (*REPEAT*);
    END (*ELSE*);
END (*IF*);

```

```

        IF ANSWER() 'YES' THEN
        BEGIN (*IF*)
            FLAG:=TRUE;
            ENDFLAG:=TRUE;
        END (*IF*)
        ELSE
        BEGIN (*ELSE*)
            WRITELN;
            WRITELN ('ENTER THE CORRECT SPELLING OF WORD. ');
            WRITE ('WORD = ');
            READLN (SPELL);
            WRITELN;
            FOR I:= (LENGTH(SPELL)) TO (SPELLENGTH-1) DO
            BEGIN (*FOR*)
                SPELL:=CONCAT(SPELL, ' ');
            END (*FOR*);
        END (*ELSE*);
    END (*IF*)
    ELSE
    BEGIN (*ELSE*)
        FLAG:=TRUE;
    END(*ELSE*);
    UNTIL FLAG;
END (*WITH*);
IF ENDFLAG=FALSE THEN
BEGIN (*IF*)
    STOREWORD;
END(*IF*);

    WRITELN;
    WRITELN('ENTER "YES" IF YOU WOULD LIKE TO ADD ');
    WRITELN('ANOTHER SPEECH FILE; ELSE ENTER "NO". ');
    WRITE('ANSWER = ');
    READLN(ANS);
    WRITELN;
    WRITELN;
    UNTIL ANS() 'YES';
END(*ELSE*);

END(*PROCEDURE STORESPEECH*);

```

{*****}

PROCEDURE WORDRECOG DESCRIPTION

THIS PROCEDURE COMPUTES A SCORE FOR EACH WORD IN THE VOCABULARY BASED ON HOW WELL THE WORD MATCHES THE GIVEN FILE. THE WORD WITH THE HIGHEST SCORE IS CHOSEN AS THE WORD THAT WAS SPOKEN. THE SPEECH FILE IS LIMITED TO ONE WORD SINCE THIS IS AN ISOLATED WORD RECOGNITION ALGORITHM. ONCE THE RECOGNITION SCORES ARE OBTAINED, THE WORD'S RECOGNITION STATISTICS ARE UPDATED (SINCE THE ACTUAL WORD SPOKEN IS STORED IN THE SPEECH FILE, THERE IS NO PROBLEM WITH UPDATING THE WRONG STATISTICS). THE FOLLOWING INFORMATION IS STORED IN FILE "RESULTS" FOR EACH SPEECH FILE EXAMINED: ACTUAL WORD SPOKEN, SPEAKER, WORD BEING SCORED, WORDSCORE, SCORECOUNT, PWORD, DELW, DELCNT, INSRTSCORE.

*****}

SEGMENT PROCEDURE WORDRECOG(WNUMB:INTEGER);

VAR

 NOWORD,I:INTEGER;
 HIGHSORE,NEXTHIGHEST:REAL;
 HIGHWORD:STRING[SPLENGTH];
 FLAG:BOOLEAN;

BEGIN(*WORDRECOG*)

 WRITELN('WORDRECOG');
 SEEK(WFILE,WNUMB);
 GET(WFILE);
 WDATA:=WFILE^;
 WORD:=WDATA.SPELLING;

 {Output automatic recognition results heading
 to file RESULTS.}

 WRITELN(RESULTS,CHR(12));
 FOR I:=1 TO 5 DO
 BEGIN (*FOR*)
 WRITELN(RESULTS);
 END(*FOR*);
 WRITELN(RESULTS,'ACTUAL WORD = ',WORD);

```

WRITELN(RESULTS);
WRITELN(RESULTS, ' :1, 'WORD', ' :15, 'WORDSCORE', ' :4, 'SCORECOUNT',
                ' :4, 'PWORD', ' :8, 'DELW', ' :8, 'DELCNT', ' :8, 'INSRTSCORE');
FOR I:=1 TO 96 DO
BEGIN(*FOR*)
    WRITE(RESULTS, '-');
END(*FOR*);
WRITELN(RESULTS);
WRITELN('WORD BEING EXAMINED = ', WORD);
SEEK(SFILE, WNUMB);
GET(SFILE);
WSTATS:=SFILE^;
IF WORDFLAG=0 THEN
BEGIN(*IF*)

    {Initialize word's recognition statistics.}

    WITH WSTATS DO
    BEGIN (*WITH*)
        NOCORRECT:=0;
        NOATTEMPT:=0;
        TOTDIFF:=0.0;
        MINDIFF:=1.0;
        TOTALSCORE:=0.0;
        SEEK(SFILE, WNUMB);
        SFILE^:=WSTATS;
        PUT (SFILE)
    END (*WITH*);
END(*IF*);
FILENAME:=CONCAT(SPEAKER, 'RECOG');
FILETITLE(SPECH, FILENAME);
RESET(SPECH);

BEGINREC:=0;
RESET(SPECH);
WHILE NOT EOF(SPECH) DO
BEGIN(*WHILE*)
    SEEK(SPECH, BEGINREC);
    GET(SPECH);
    IOUT:=SPECH^;
    IF IOUT.SPELL=WORD THEN
    BEGIN(*IF*)
        IF WORDCNT=WORDFLAG THEN
        BEGIN(*IF*)
            WRITELN(RESULTS);
            WRITELN(RESULTS, 'NEW SPEECH FILE: ', 'SPEAKER = ', IOUT.SPKR);
            HIGHSCORE:=0.0;
            HIGHWORD:='XXXXX';
            NEXTHIGHEST:=0.0;
            RESET(WFILE);
            NOWORD:=0;
            WHILE NOT EOF(WFILE) DO
            BEGIN(*WHILE*)
                SEEK(WFILE, NOWORD);

```

```

GET(WFILE);
WDATA:=WFILE^;
WORD:=WDATA.SPELLING;

{Output recognition information to screen, and
to file RESULTS.}

WRITELN;
WRITELN('-----');
WRITELN('WORD = ',IOUT.SPELL,' CONTINUATION # = ',WNUMB);
WRITELN('WORD COUNT = ',WORDCNT);
WRITELN('WORD ATTEMPTED = ',WORD);
GETWORDVARS;
SCOREWORD;
WRITELN(RESULTS,WORD,WORDSCORE:8,SCORECOUNT:11,' ':7,
        PWORD:6:2,' ':6,DELW:6:2,DELCNT:12,' ':11,INSRTSCORE:6:2);
WRITELN('WORDSCORE = ',WORDSCORE);
WRITELN('-----');
IF NOWORD=WNUMB THEN
BEGIN(*IF*)
    SEEK(SFILE,WNUMB);
    GET(SFILE);
    WSTATS:=SFILE^;
    WSTATS.TOTALSCORE:=WSTATS.TOTALSCORE+WORDSCORE;
    SEEK(SFILE,WNUMB);
    SFILE^:=WSTATS;
    PUT(SFILE);
END(*IF*);

{Determine highest scoring word, and score of
word with second highest score.}

IF (WORDSCORE=HIGHSORE)AND(WORDSCORE>NEXTHIGHEST) THEN
BEGIN(*IF*)
    NEXTHIGHEST:=WORDSCORE;
END(*IF*);
IF ((WORDSCORE)>HIGHSORE) THEN
BEGIN(*IF*)
    NEXTHIGHEST:=HIGHSORE;
    HIGHSORE:=WORDSCORE;
    HIGHWORD:=WORD;
END(*IF*);
SEEK(WFILE,NOWORD);
GET(WFILE);
GET(WFILE);
NOWORD:=NOWORD+1;
END (*WHILE*);
SEEK(WFILE,WNUMB);
GET(WFILE);
WDATA:=WFILE^;
WORD:=WDATA.SPELLING;

{Update word's recognition statistics.}

```

```

        SEEK(SFILE,WNUMB);
        GET(SFILE);
        WSTATS:=SFILE^;
        WSTATS.NOATTEMPT:=WSTATS.NOATTEMPT+1;
        IF HIGHWORD=WDATA.SPELLING THEN
        BEGIN(*IF*)
            WSTATS.NOCORRECT:=WSTATS.NOCORRECT+1;
            WSTATS.TOTDIFF:=WSTATS.TOTDIFF+(HIGHSCORE-NEXTHIGHEST);
            IF (HIGHSCORE-NEXTHIGHEST)<WSTATS.MINDIFF THEN
            BEGIN(*IF*)
                WSTATS.MINDIFF:=HIGHSCORE-NEXTHIGHEST;
            END (*IF*);
        END(*IF*);
        SEEK(SFILE,WNUMB);
        SFILE^:=WSTATS;
        PUT(SFILE);
    END(*IF*);
    FLAG:=TRUE;
    WHILE ((FLAG=TRUE)OR(IOUT.CONTFLAG=1))DO
    BEGIN (*WHILE*)
        FLAG:=FALSE;
        SEEK(SPECH,BEGINREC);
        GET(SPECH);
        IOUT:=SPECH^;
        BEGINREC:=BEGINREC+1;
    END (*WHILE*);
    GET(SPECH);
    WORDCNT:=WORDCNT+1;

    END(*IF*)
    ELSE
    BEGIN(*ELSE*)
        BEGINREC:=BEGINREC+1;
        GET(SPECH);
    END(*ELSE*);
    END(*WHILE*);

    CLOSE(SPECH);

    END (*PROCEDURE WORDRECOG*);

```

{*****}

PROCEDURE FUZZYOPT DESCRIPTION

THIS PROCEDURE ATTEMPTS TO OPTIMIZE THE SPECIFIED WORD
FUZZY VARIABLES. THE OPTIMIZATION IS ACCOMPLISHED BY FIRST ALTERING

A FUZZY VARIABLE WITHIN ITS SPECIFIED LIMITS, COMPUTING THE
RECOGNITION RESULTS, AND THEN ADJUSTING THE VARIABLE'S VALUE AND
LIMITS BASED ON THESE RESULTS.

*****}

SEGMENT PROCEDURE FUZZYOPT(NOPTIONS:INTEGER);

VAR

QUIT:BOOLEAN;
NOWORD,I,FINT:INTEGER;
SAVEVAR,SAVESCORE:REAL;
RECSCR1,RECSCR2:REAL;
MINSORE:REAL;
OPTANSWR:ARRAY[1..NOFUZZYVARS] OF STRING[3];

{*****}

PROCEDURE GETOPTVARS DESCRIPTION

THIS PROCEDURE INITIALIZES THE VALUES OF THE GLOBAL FUZZY
VARIABLES TO CORRESPOND TO A WORD'S OPTIMUM FUZZY VARIABLES,
WHICH ARE STORED IN FILE FZOPT. THE VALUE OF NOWORD INDICATES
THE APPROPRIATE WORD.

*****}

PROCEDURE GETOPTVARS(NOWORD:INTEGER);

BEGIN(*GETOPTVARS*)

SEEK(FZOPT,NOWORD);
GET(FZOPT);
FLIMIT:=FZOPT^;

```

WITH FLIMIT DO
BEGIN(*WITH*)
  STHR:=FUZVAR[1];
  SUBE:=FUZVAR[2];
  SUBF:=FUZVAR[3];
  INSE:=FUZVAR[4];
  INSF:=FUZVAR[5];
  DELE:=FUZVAR[6];
  DELF:=FUZVAR[7];
  DELG:=FUZVAR[8];
  DCNE:=FUZVAR[9];
  DCNF:=FUZVAR[10];
  DCNG:=FUZVAR[11];
  SFE:=FUZVAR[12];
  SFF:=FUZVAR[13];
  CHVE:=FUZVAR[14];
  CHVF:=FUZVAR[15];
  STATE:=FUZVAR[16];
  STATF:=FUZVAR[17];
  STATG:=FUZVAR[18];
  THR1E:=FUZVAR[19];
  THR1F:=FUZVAR[20];
  THR2E:=FUZVAR[21];
  THR2F:=FUZVAR[22];
END(*WITH*);

```

```

END(*PROCEDURE GETOPTVARS*);

```

```

{*****}

```

PROCEDURE RECOGSCR DESCRIPTION -----

THIS PROCEDURE COMPUTES A FIGURE OF MERIT USED IN DETERMINING WHETHER OR NOT A CHANGE IN A FUZZY VARIABLE CORRESPONDED TO AN INCREASE IN THE RECOGNITION ACCURACY. THIS FIGURE IS CALCULATED USING THE RECOGNITION RESULTS STORED IN FILE SFILF.

```

{*****}

```

```
PROCEDURE RECOGSCR(VAR RECSCR:REAL; WNUMB:INTEGER);
```

```
CONST
```

```
  FV1=0.6;  
  FV2=0.25;  
  FV3=0.05;  
  FV4=0.10;
```

```
BEGIN(*RECOGSCR*)
```

```
  WRITELN('RECOGSCR');
```

```
  SEEK(SFILE,WNUMB);
```

```
  GET(SFILE);
```

```
  WSTATS:=SFILE^;
```

```
  WITH WSTATS DO
```

```
  BEGIN(*WITH*)
```

```
    IF (NOCORRECT<>0) THEN
```

```
    BEGIN(*IF*)
```

```
      RECSCR:=FV1*(NOCORRECT/NOATTEMPT)+FV2*(TOTDIFF/NOCORRECT)+  
              FV3*(MINDIFF)+FV4*(TOTALSCORE/NOATTEMPT);
```

```
    END(*IF*)
```

```
  ELSE
```

```
  BEGIN(*ELSE*)
```

```
    RECSCR:=0;
```

```
  END(*ELSE*);
```

```
END(*WITH*);
```

```
END(*PROCEDURE RECOGSCR*);
```

```
{*****}
```

```
PROCEDURE GETLIMITS DESCRIPTION
```

```
-----
```

THIS PROCEDURE RETRIEVES THE INITIAL FUZZY VARIABLE VALUES,
MAXIMUM LIMITS, AND MINIMUM LIMITS WHICH ARE USED TO INITIALIZE
A NEW WORD'S OPTIMUM FUZZY VARIABLES.

```
*****}
```

PROCEDURE GETLIMITS;

VAR

I:INTEGER;

BEGIN(*GETLIMITS*)

RESET(OPTFUZZY);

WITH FLIMIT DO

BEGIN(*WITH*)

FOR I:=1 TO NOFUZZYVARS DO

BEGIN(*FOR*)

READLN(OPTFUZZY,FUZVAR[I],MAXLIMIT[I],MINLIMIT[I],FUZNAME[I]);

END(*FOR*);

RECScore:=0;

END(*WITH*);

CLOSE(OPTFUZZY);

END(*GETLIMITS*);

{*****}

PROCEDURE SETLIMITS DESCRIPTION

THIS PROCEDURE INITIALIZES THE OPTIMUM FUZZY VARIABLES FOR
ALL WORD'S WHOSE VARIABLES WERE NOT PREVIOUSLY INITIALIZED.

{*****}

PROCEDURE SETLIMITS;

VAR

I,WORDCNT:INTEGER;

BEGIN(*SETLIMITS*)

RESET(WFILE);

NOWORD:=0;

RESET(FZOPT);

WHILE NOT EOF(FZOPT) DO

BEGIN(*WHILE*)

SEEK(FZOPT,NOWORD);

GET(FZOPT);

```

        GET(FZOPT);
        NOWORD:=NOWORD+1;
    END(*WHILE*);
    NOWORD:=NOWORD-1;
    WORDCNT:=0;
    RESET(WFILE);
    WHILE NOT EOF(WFILE) DO
    BEGIN(*WHILE*)
        SEEK(WFILE,WORDCNT);
        GET(WFILE);
        GET(WFILE);
        WORDCNT:=WORDCNT+1;
    END(*WHILE*);
    WORDCNT:=WORDCNT-1;

    IF WORDCNT>NOWORD THEN
    BEGIN(*IF*)
        IF NOWORD>-1 THEN
        BEGIN(*IF*)
            SEEK(FZOPT,(NOWORD));
            GET(FZOPT);
        END(*IF*);
        FOR I:=(NOWORD+1) TO WORDCNT DO
        BEGIN(*FOR*)
            GETLIMITS;
            FZOPT^:=FLIMIT;
            PUT(FZOPT);
            SEEK(FZOPT,I);
            GET(FZOPT);
        END(*FOR*);
    END(*IF*);

END(*SETLIMITS*);

```

{*****}

PROCEDURE OPTRECOG DESCRIPTION

THIS PROCEDURE IS IDENTICAL TO PROCEDURE WORDRECOG, EXCEPT THAT
NO RESULTS ARE STORED.

{*****}

```
PROCEDURE OPTRECOG(WNUMB:INTEGER);
```

```
VAR
```

```
  NOWORD,I:INTEGER;  
  HIGHSORE,NEXTHIGHEST:REAL;  
  HIGHWORD:STRING[SPELENGTH];  
  FLAG:BOOLEAN;
```

```
BEGIN(*OPTRECOG*)
```

```
  WRITELN('OPTRECOG');  
  SEEK(WFILE,WNUMB);  
  GET(WFILE);  
  WDATA:=WFILE^;  
  WORD:=WDATA.SPELLING;  
  WRITELN('WORD BEING EXAMINED = ',WORD);  
  SEEK(SFILE,WNUMB);  
  GET(SFILE);  
  WSTATS:=SFILE^;
```

```
  {Initialize word's recognition statistics.}
```

```
  WITH WSTATS DO
```

```
  BEGIN (*WITH*)
```

```
    NOCORRECT:=0;  
    NOATTEMPT:=0;  
    TOTDIFF:=0.0;  
    MINDIFF:=1.0;  
    TOTALSCORE:=0.0;  
    SEEK(SFILE,WNUMB);  
    SFILE^:=WSTATS;  
    PUT (SFILE)
```

```
  END (*WITH*);
```

```
  FILENAME:=CONCAT(SPEAKER,'RECOG');
```

```
  FILETITLE(SPECH,FILENAME);
```

```
  RESET(SPECH);
```

```
  BEGINREC:=0;
```

```
  RESET(SPECH);
```

```
  WHILE NOT EOF(SPECH) DO
```

```
  BEGIN(*WHILE*)
```

```
    SEEK(SPECH,BEGINREC);
```

```
    GET(SPECH);
```

```
    IOUT:=SPECH^;
```

```
    IF IOUT.SPELL=WORD THEN
```

```
    BEGIN(*IF*)
```

```
      HIGHSORE:=0.0;
```

```
      HIGHWORD:='XXXXX';
```

```
      NEXTHIGHEST:=0.0;
```

```
      RESET(WFILE);
```

```

NOWORD:=0;
WHILE NOT EOF(WFILE) DO
BEGIN(*WHILE*)
    SEEK(WFILE,NOWORD);
    GET(WFILE);
    WDATA:=WFILE^;
    WORD:=WDATA.SPELLING;
    WRITELN;
    WRITELN('-----');
    WRITELN('WORD = ',IOUT.SPELL,' CONTINUATION # = ',WNUMB);
    WRITELN('WORD ATTEMPTED = ',WORD);
    GETOPTVARS(NOWORD);
    SCOREWORD;
    WRITELN('WORDSCORE = ',WORDSCORE);
    WRITELN('-----');
    IF NOWORD=WNUMB THEN
    BEGIN(*IF*)
        SEEK(SFILE,WNUMB);
        GET(SFILE);
        WSTATS:=SFILE^;
        WSTATS.TOTALSCORE:=WSTATS.TOTALSCORE+WORDSCORE;
        SEEK(SFILE,WNUMB);
        SFILE^:=WSTATS;
        PUT(SFILE);
    END(*IF*);

    {Determine highest scoring word, and score of the
    second highest word score.}

    IF (WORDSCORE<=HIGHSCORE)AND(WORDSCORE>NEXTHIGHEST) THEN
    BEGIN(*IF*)
        NEXTHIGHEST:=WORDSCORE;
    END(*IF*);
    IF ((WORDSCORE)>HIGHSCORE) THEN
    BEGIN(*IF*)
        NEXTHIGHEST:=HIGHSCORE;
        HIGHSCORE:=WORDSCORE;
        HIGHWORD:=WORD;
    END(*IF*);
    SEEK(WFILE,NOWORD);
    GET(WFILE);
    GET(WFILE);
    NOWORD:=NOWORD+1;
END (*WHILE*);
FLAG:=TRUE;
WHILE ((FLAG=TRUE)OR(IOUT.CONTFLAG=1))DO
BEGIN (*WHILE*)
    FLAG:=FALSE;
    SEEK(SPECH,BEGINREC);
    GET(SPECH);
    IOUT:=SPECH^;
    BEGINREC:=BEGINREC+1;
END (*WHILE*);
GET(SPECH);

```

```

    SEEK(WFILE,WNUMB);
    GET(WFILE);
    WDATA:=WFILE^;
    WORD:=WDATA.SPELLING;

    {Update word's recognition statistics.}

    SEEK(SFILE,WNUMB);
    GET(SFILE);
    WSTATS:=SFILE^;
    WSTATS.NOATTEMPT:=WSTATS.NOATTEMPT+1;
    IF HIGHWORD=WDATA.SPELLING THEN
    BEGIN(*IF*)
        WSTATS.NOCORRECT:=WSTATS.NOCORRECT+1;
        WSTATS.TOTDIFF:=WSTATS.TOTDIFF+(HIGHSCORE-NEXTHIGHEST);
        IF (HIGHSCORE-NEXTHIGHEST)<WSTATS.MINDIFF THEN
        BEGIN(*IF*)
            WSTATS.MINDIFF:=HIGHSCORE-NEXTHIGHEST;
        END (*IF*);
    END(*IF*);
    SEEK(SFILE,WNUMB);
    SFILE^:=WSTATS;
    PUT(SFILE);

    END(*IF*)
    ELSE
    BEGIN(*ELSE*)
        BEGINREC:=BEGINREC+1;
        GET(SPECH);
    END(*ELSE*);
    END(*WHILE*);

    CLOSE(SPECH);

    END (*PROCEDURE OPTRECOG*);

PROCEDURE AUTORECOG(VAR RECSCR:REAL);

BEGIN
    SEEK(FZOPT,WNUMB);
    FZOPT^:=FLIMIT;
    PUT(FZOPT);
    OPTRECOG(WNUMB);
    RECOGSCR(RECSCR,WNUMB);
    SEEK(FZOPT,WNUMB);
    GET(FZOPT);
    FLIMIT:=FZOPT^;
    END(*PROCEDURE AUTORECOG*);

```

```

BEGIN(*FUZZYOPT*)

  IF NOPTIONS=1 THEN
    BEGIN(*IF*)
      GETLIMITS;

      {Get names of fuzzy variables to be optimized.}

      WRITELN;
      WRITELN('ENTER "YES" NEXT TO THE FUZZY VARIABLE');
      WRITELN('NAMES TO BE OPTIMIZED. ');
      FOR I:=1 TO NOFUZZYVARS DO
        BEGIN(*FOR*)
          WRITE(FLIMIT.FUZNAME[I], '? : ');
          READLN(OPTANSWR[I]);
        END(*FOR*);
      WRITELN;
    END(*IF*)
  ELSE
    BEGIN(*ELSE*)
      FOR I:=1 TO NOFUZZYVARS DO
        BEGIN(*FOR*)
          OPTANSWR[I]:='YES';
        END(*FOR*);
      END(*ELSE*);
      SETLIMITS;

      RESET(FZOPT);
      RESET(SFILE);
      RESET(WFILE);
      WNUMB:=0;
      WRITELN;
      SEEK(WFILE, WNUMB);
      GET(WFILE);

      {Compute initial recognition statistics for
      each word being examined.}

      WHILE NOT EOF(WFILE) DO
        BEGIN(*WHILE*)
          IF OPERATEC(WNUMB+1)='YES' THEN
            BEGIN(*IF*)
              OPTRECOG(WNUMB);
            END(*IF*);
            SEEK(WFILE, WNUMB);
            GET(WFILE);
            GET(WFILE);
            WNUMB:=WNUMB+1;
          END(*WHILE*);

          QUIT:=FALSE;
          REPEAT
            FOR FINI:=1 TO NOFUZZYVARS DO
              BEGIN(*FOR*)

```

```

IF OPTANSWR[FINT]='YES' THEN
BEGIN(*IF*)
  WNUMB:=0;
  RESET(WFILE);
  WHILE NOT EOF(WFILE) DO
  BEGIN(*WHILE*)
    IF OPERATE[WNUMB+1]='YES' THEN
    BEGIN(*IF*)

      {Compute initial Figure of Merit for word.}

      RECOGSCR(SAVESCORE,WNUMB);
      RECSCR1:=SAVESCORE;
      GETOPTVARS(WNUMB);

      {Change next fuzzy variable to be evaluated,
      and adjust its value and limits based on the word's
      recognition statistics using the variables new value.}

      WITH FLIMIT DO
      BEGIN(*WITH*)
        SAVEVAR:=FUZVAR[FINT];
        FUZVAR[FINT]:=(FUZVAR[FINT]+MINLIMIT[FINT])/2;
        AUTORECOG(RECSCR2);
        IF RECSCR2>RECSCR1 THEN
        BEGIN(*IF*)
          MAXLIMIT[FINT]:=SAVEVAR;
          RECSCR1:=RECSCR2;
        END(*IF*)
        ELSE
        BEGIN(*ELSE*)
          MINLIMIT[FINT]:=FUZVAR[FINT];
          IF SAVEVAR<(MAXLIMIT[FINT]-0.5) THEN
          BEGIN(*IF*)
            FUZVAR[FINT]:=SAVEVAR+0.5;
          END(*IF*)
          ELSE
          BEGIN(*ELSE*)
            FUZVAR[FINT]:=(SAVEVAR+MAXLIMIT[FINT])/2;
          END(*ELSE*);
          AUTORECOG(RECSCR2);
          IF (RECSCR2>RECSCR1) THEN
          BEGIN(*IF*)
            MINLIMIT[FINT]:=SAVEVAR;
            RECSCR1:=RECSCR2;
          END(*IF*)
          ELSE
          BEGIN(*ELSE*)
            MAXLIMIT[FINT]:=FUZVAR[FINT];
            FUZVAR[FINT]:=SAVEVAR;
          END(*ELSE*);
        END(*ELSE*);
        RECSCORE:=RECSCR1;
        SEEK(FZOPT,WNUMB);

```

```

        FZOPT^:=FLIMIT;
        PUT(FZOPT);
        END(*WITH*);

    END(*IF*);

    SEEK(WFILE,WNUMB);
    GET(WFILE);
    GET(WFILE);
    WNUMB:=WNUMB+1;
    END(*WHILE*);
    END(*IF*);

    END(*FOR*);
    UNTIL (QUIT=TRUE);
    END(*PROCEDURE FUZZYOPT*);

```

{*****}

PROCEDURE AUTOMAT DESCRIPTION

THIS PROCEDURE PROVIDES THE FOLLOWING OPTIONS:

0. EXECUTE UP TO "MAXNOPTIONS" OF THE FOLLOWING OPTIONS.
(OPTIONS 2, 4, 5, AND 6 DEFAULT TO OPERATE ON ALL WORDS,
AND OPTION 6 DEFAULTS TO OPERATE ON ALL VARIABLES WHEN THIS
OPTION IS USED. THE REMAINING OPTIONS HAVE NO DEFAULT
VALUES, AND REQUIRE USER INPUT BEFORE THEY CAN BE EXECUTED.)
1. STORE A NUMBER OF SPEECH FILES FOR FUTURE USE.
2. COMPUTE THE WORD RECOGNITION RESULTS FOR A SET OF SPECIFIED
WORDS, USING THE SPEECH FILES STORED IN FILE RECOG.
3. COMPUTE A GIVEN WORD'S RECOGNITION RESULTS USING THE SPEECH
FILES STORED IN FILE RECOG.
4. INITIALIZE THE WORD STATISTICS FOR A SET OF SPECIFIED
WORDS, USING THE SPEECH FILES STORED IN FILE SPECH.
5. DETERMINE THE PHONEME REPRESENTATIONS FOR A SET OF SPECIFIED
WORDS, USING THE SPEECH FILES STORED IN FILE SPECH.
6. OPTIMIZE THE SPECIFIED FUZZY VARIABLES FOR A SET OF
SPECIFIED WORDS, USING THE SPEECH FILES STORED IN FILE RECOG.
7. CHANGE THE OPTIMUM FUZZY VARIABLES FOR A GIVEN WORD, OR
CHANGE THE INITIAL OPTIMUM FUZZY VARIABLES STORED IN
FILE OPTFUZZY.

PROCEDURE CONSTANTS

CONSTANTS

DESCRIPTION

MAXNOPTIONS - THE MAXIMUM NUMBER OF OPTIONS THAT CAN BE
EXECUTED CONSECUTIVELY.

*****}

SEGMENT PROCEDURE AUTOMAT;

CONST

MAXNOPTIONS=4;

VAR

I,NOPTIONS:INTEGER;
AOPTION:ARRAY[1..MAXNOPTIONS] OF INTEGER;
NOWORD:INTEGER;
STRRECORD:INTEGER;

{*****}

PROCEDURE GETOPWORDS DESCRIPTION

THIS PROCEDURE ALLOWS THE USER TO SPECIFY THE WORDS THAT ARE
TO BE OPERATED ON.

*****}

PROCEDURE GETOPWORDS;

VAR

I,WORDCNT:INTEGER;
ANSWER:STRING[3];

BEGIN(*GETOPWORDS*)

```
WORDCNT:=1;
ANSWER:='YES';
IF NOPTIONS=1 THEN
BEGIN(*IF*)
  WRITELN('ENTER "YES" NEXT TO THE WORDS TO');
  WRITELN('BE OPERATED ON. ');
  WRITELN;
  WRITE('OPERATE ON ALL WORDS? : ');
  READLN(ANSWER);
  IF ANSWER<>'YES' THEN
  BEGIN(*IF*)
    WORDCNT:=0;
    RESET(WFILE);
    WHILE NOT EOF(WFILE) DO
    BEGIN(*WHILE*)
      SEEK(WFILE,WORDCNT);
      GET(WFILE);
      WDATA:=WFILE^;
      WRITE(WDATA.SPELLING,'? : ');
      READLN(OPERATE[WORDCNT+1]);
      GET(WFILE);
      WORDCNT:=WORDCNT+1;
    END(*WHILE*);
    WORDCNT:=WORDCNT+1;
  END(*IF*);
END(*IF*);
FOR I:=WORDCNT TO MAXNOWORDS DO
BEGIN(*FOR*)
  OPERATE[I]:=ANSWER;
END(*FOR*);
```

END(*PROCEDURE GETOPWORDS*);

(*****)

PROCEDURE INITALL DESCRIPTION

THIS PROCEDURE INITIALIZES THE WORD STATISTICS FOR A SET OF SPECIFIED WORDS, USING THE SPEECH FILES STORED IN FILE SPECH. IF OPTION 0 IS USED, THE WORD STATISTICS FOR ALL THE WORDS IN THE VOCABULARY WILL BE INITIALIZED.

(*****)

```

PROCEDURE INITALL;
VAR
    J,INT:INTEGER;
BEGIN(*INITALL*)
    WRITELN('INITALL');
    IF NOPTIONS=1 THEN
        INITFILE;
        GETOPWORDS;
        GETFUZZYVARS;
        RESET(SFILE);
        RESET(WFILE);
        NOWORD:=0;
        WHILE NOT EOF(WFILE) DO
            BEGIN(*WHILE*)
                IF OPERATEC[NOWORD+1]='YES' THEN
                    BEGIN(*IF*)
                        INITSTAT(NOWORD);
                    END(*IF*);
                    SEEK(WFILE,NOWORD);
                    GET(WFILE);
                    GET(WFILE);
                    NOWORD:=NOWORD+1;
                END(*WHILE*);
            END(*PROCEDURE INITALL*);

```

{*****}

PROCEDURE RECOGWORD DESCRIPTION

THIS PROCEDURE COMPUTES A GIVEN WORD'S RECOGNITION RESULTS
USING THE SPEECH FILES STORED IN FILE RECOG.

*****}

PROCEDURE RECOGWORD;

```

VAR
  I:INTEGER;
  TEMPWORD:STRING[SPELLENGTH];

BEGIN(*RECOGWORD*)

  INITFILE;
  WRITELN;
  WRITELN ('INPUT EXACT SPELLING OF WORD TO BE RECOGNIZED ');
  WRITE ('WORD = ');
  READLN (WORD);
  FOR I:=(LENGTH(WORD)) TO (SPELLENGTH-1) DO
    BEGIN(*FOR*)
      WORD:=CONCAT(WORD,' ');
    END(*FOR*);
  WRITELN ;
  RESET(SFILE);
  RESET (WFILE);
  TEMPWORD:=' ';
  WNUMB:=0;
  WORDFLAG:=0;
  WORDCNT:=0;
  WHILE (NOT EOF(WFILE))AND(WORD<>TEMPWORD) DO
    BEGIN (*WHILE*)
      SEEK(WFILE,WNUMB);
      GET(WFILE);
      WDATA:=WFILE^;
      GET(WFILE);
      TEMPWORD:=WDATA.SPELLING;
      WNUMB:=WNUMB+ 1
    END (*WHILE*);
    WNUMB:=WNUMB-1;
    IF (WORD<>TEMPWORD) THEN
      BEGIN (*IF*)
        WRITELN;
        WRITELN ('WORD DOES NOT EXIST. ');
      END (*IF*)
    ELSE
      BEGIN (*ELSE*)
        REWRITE(RESULTS);
        WORDRECOG(WNUMB);
        CLOSE(RESULTS);
      END(*ELSE*);
      WRITELN('THE RECOGNITION RESULTS ARE STORED IN "RESULTS".');
    END(*PROCEDURE RECOGWORD*);

```

(*****)

PROCEDURE RECOGALL DESCRIPTION

THIS PROCEDURE COMPUTES THE RECOGNITION RESULTS FOR A SET OF SPECIFIED WORDS, USING THE SPEECH FILES STORED IN FILE RECOG. IF OPTION 0 IS NOT USED, THIS PROCEDURE CAN BE RESTARTED AT A GIVEN CONTINUATION NUMBER.

(*****)

PROCEDURE RECOGALL;

BEGIN(*RECOGALL*)

```
WORDFLAG:=0;
WORDCNT:=0;
IF NOPTIONS=1 THEN
  BEGIN(*IF*)
    INITFILE;
    WRITELN('IF YOU ARE CONTINUING A PREVIOUS RECOGNITION RUN:');
    WRITELN;
    WRITELN('ENTER THE "CONTINUATION NUMBER"; ELSE ');
    WRITELN('ENTER "0".');
    WRITE('CONTINUATION * = ');
    READLN(WNUMB);
    WRITELN;
    WRITELN('ENTER THE "WORD COUNT"; ELSE ENTER "0".');
    WRITE('WORD COUNT = ');
    READLN(WORDFLAG);
    WRITELN;
  END(*IF*)
ELSE
  BEGIN(*ELSE*)
    WNUMB:=0;
  END(*ELSE*);
  GETOPWORDS;
  RESET(WFILE);
  RESET(SFILE);
  IF (WORDFLAG=0) AND (WNUMB=0) THEN
    BEGIN(*IF*)
      REWRITE(RESULTS);
    END(*IF*)
  ELSE
    BEGIN(*ELSE*)
      APPEND(RESULTS);
    END(*ELSE*);
  SEEK(WFILE,WNUMB);
```

```

GET(WFILE);
WHILE NOT EOF(WFILE) DO
BEGIN(*WHILE*)
  IF OPERATE(WNUMB+1)='YES' THEN
  BEGIN(*IF*)
    WORDRECOG(WNUMB);
  END(*IF*);
  SEEK(WFILE,WNUMB);
  GET(WFILE);
  GET(WFILE);
  WORDFLAG:=0;
  WORDCNT:=0;
  WNUMB:=WNUMB+1;
END(*WHILE*);
CLOSE(RESULTS);
WRITELN('THE RECOGNITION RESULTS ARE STORED IN "RESULTS".');

END(*PROCEDURE RECOGALL*);

```

(*****)

PROCEDURE DETPHALL DESCRIPTION

THIS PROCEDURE DETERMINES THE PHONEME REPRESENTATIONS FOR A SET OF SPECIFIED WORDS, USING THE SPEECH FILES STORED IN FILE SPECH. IF OPTION 0 IS NOT USED, THE USER MUST ENTER THE WORDS TO BE OPERATED ON, THE NUMBER OF VECTORS PER WORD PHONEME, AND THE MAXIMUM NUMBER OF ITERATIONS (NOITERATIONS) TO BE PERFORMED. ALSO, THIS PROCEDURE CAN BE RESTARTED AT A GIVEN CONTINUATION NUMBER. HOWEVER, IF OPTION 0 IS USED, THE PHONEME REPRESENTATION OF ALL THE WORDS IS DETERMINED, THE NUMBER OF PHONEME VECTORS PER WORD (NOPHVECTORS) DEFAULTS TO THE VALUE OF "NOTOAVG", AND THE VALUE OF NOITERATIONS DEFAULTS TO 2.

(*****)

PROCEDURE DETPHALL;

```

VAR
  NOPHVECTORS:INTEGER;
  NOITERATIONS,ICOUNT:INTEGER;

```

BEGIN(*DETPHALL*)

IF NOPTIONS=1 THEN

BEGIN(*IF*)

INITFILE;

WRITE('IF YOU ARE CONTINUING A PREVIOUS RECOGNITION');

WRITE('RUN ENTER THE "CONTINUATION NUMBER"; ELSE ');

WRITE('ENTER "0".');

WRITE('CONTINUATION # = ');

READLN(NOWORD);

WRITELN;

WRITELN;

WRITE('ENTER THE NUMBER OF VECTORS PER WORD PHONEME.');

WRITE('# VECTORS = ');

READLN(NOPHVECTORS);

WRITELN;

WRITE('ENTER THE NUMBER OF ITERATIONS TO BE PERFORMED.');

WRITE('NO. ITERATIONS = ');

READLN(NOITERATIONS);

WRITELN;

END(*IF*)

ELSE

BEGIN(*ELSE*)

NOWORD:=0;

IF NOTOAVG = 1 THEN

BEGIN(*IF*)

NOPHVECTORS:=1;

END(*IF*)

ELSE

BEGIN(*ELSE*)

NOPHVECTORS:=NOTOAVG;

END(*ELSE*);

NOITERATIONS:=2;

END(*ELSE*);

GETOPWORDS;

RESET(SFILE);

RESET(WFILE);

FILETITLE(STATSOUT,'PHREPDAT');

REWRITE(STATSOUT);

GETFUZZYVARS;

SEEK(WFILE,NOWORD);

GET(WFILE);

WHILE NOT EOF(WFILE) DO

BEGIN(*WHILE*)

IF OPERATEL[NOWORD+1]='YES' THEN

BEGIN(*IF*)

WRITE('DETPHALL CONTINUATION NUMBER= ',NOWORD);

WRITELN;

OPERATION:='AUTODET';

AUTODET(NOWORD,NOPHVECTORS,CHNGEFLAG);

INITSTAT(NOWORD);

ICOUNT:=0;

REPEAT

ICOUNT:=ICOUNT+1;

```

        CHNGEFLAG:=FALSE;
        OPERATION:='CORRECTR';
        AUTODET(NOWORD,NOPHVECTORS,CHNGEFLAG);
        INITSTAT(NOWORD);
        OPERATION:='ADJUSTDE';
        AUTODET(NOWORD,NOPHVECTORS,CHNGEFLAG);
        INITSTAT(NOWORD);
    UNTIL (ICOUNT=NOITERATIONS)OR(CHNGEFLAG=FALSE);
END(*IF*);
SEEK(WFILE,NOWORD);
GET(WFILE);
GET(WFILE);
NOWORD:=NOWORD+1;
END(*WHILE*);
CLOSE(STATSOUT);

END(*PROCEDURE DETPHALL*);

BEGIN (*AUTOMAT*)

    OPERATION:='-----';
    WRITELN;
    WRITELN;
    WRITELN('CHOOSE OPTION FROM FOLLOWING LIST');
    WRITELN;
    WRITELN('0 - EXECUTE UP TO ',MAXNOPTIONS,' CONSECUTIVE OPTIONS. ');
    WRITELN('1 - STORE SPEECH FILES. ');
    WRITELN('2 - AUTOMATIC WORD RECOGNITION FOR MULTIPLE WORDS. ');
    WRITELN('3 - AUTOMATIC WORD RECOGNITION FOR A GIVEN WORD. ');
    WRITELN('4 - INITIALIZE MULTIPLE WORD STATISTICS. ');
    WRITELN('5 - DETERMINE MULTIPLE WORD PHONEME REPRESENTATIONS. ');
    WRITELN('6 - OPTIMIZE THE FUZZY VARIABLES. ');
    WRITELN('7 - CHANGE THE OPTIMUM FUZZY VARIABLES. ');
    WRITELN;
    WRITE('OPTION = ');
    READLN(AOPTION[1]);
    WRITELN;
    NOPTIONS:=1;

    IF AOPTION[1] = 0 THEN
        BEGIN(*IF*)
            REPEAT
                WRITELN('ENTER OPTION #',NOPTIONS,'; OR ENTER 0. ');
                WRITE('OPTION[' ,NOPTIONS,'] = ');
                READLN(AOPTION[NOPTIONS]);
                NOPTIONS:=NOPTIONS+1;
            UNTIL ((AOPTION[NOPTIONS-1]=0)OR(NOPTIONS=(MAXNOPTIONS+1)));
            NOPTIONS:=NOPTIONS-1;
            INITFILE;
        END(*IF*);
        FOR I:=1 TO NOPTIONS DO

```

```

BEGIN(*FOR*)
  IF AOPTION[I] = 1 THEN
    BEGIN(*IF*)
      STORESPEECH;
    END(*IF*)
  ELSE IF AOPTION[I] = 2 THEN
    BEGIN(*ELSE*)
      RECOGALL;
    END(*ELSE*)
  ELSE IF AOPTION[I] = 3 THEN
    BEGIN(*ELSE*)
      RECOGWORD;
    END(*ELSE*)
  ELSE IF AOPTION[I] = 4 THEN
    BEGIN(*ELSE*)
      INITALL;
    END(*ELSE*)
  ELSE IF AOPTION[I] = 5 THEN
    BEGIN(*ELSE*)
      DETPHALL;
    END(*ELSE*)
  ELSE IF AOPTION[I] = 6 THEN
    BEGIN(*ELSE*)
      IF NOPTIONS=1 THEN
        BEGIN(*IF*)
          INITFILE;
        END(*IF*);
        GETOPWORDS;
        FUZZYOPT(NOPTIONS);
      END(*ELSE*)
    ELSE IF AOPTION[I] = 7 THEN
      BEGIN(*ELSE*)
        INITFILE;
        CHNGEOPT;
      END(*ELSE*)
    ELSE
      BEGIN(*ELSE*)
        WRITELN;
        WRITELN('OPTION DOES NOT EXIST');
        WRITELN;
      END(*ELSE*);
      CLOSE(OPTFUZZY);
      CLOSE(FZOPT);
      CLOSE(WFILE);
      CLOSE(SFILE);
      CLOSE(SPECH);
    END(*ELSE*);
  END(*FOR*);

```

```

END (*PROCEDURE AUTOMAT*);

```

{*****}

PROGRAM OUTSTAT DESCRIPTION

THIS PROGRAM OUTPUTS ALL THE INFORMATION STORED IN FILES SFILE AND WFILE FOR THE SPECIFIED WORD (OR WORDS). THE FOLLOWING TWO OPTIONS ARE PROVIDED:

1. OUTPUT THE DATA STORED IN FILES SFILE AND WFILE FOR A GIVEN WORD TO THE FILE INDICATED BY THE USER.
2. OUTPUT THE DATA STORED IN FILES SFILE AND WFILE FOR ALL THE WORDS IN THE VOCABULARY, AND GIVE EACH OUTPUT FILE THE NAME DERIVED FROM THE WORD'S SPELLING PRECEDED BY AN "S". FOR EXAMPLE, THE OUTPUT FILE NAME FOR THE WORD ZERO WOULD BE "SZERO".

THIS PROGRAM ALSO STORES THE OVERALL FUZZY VARIABLES, AND THE WORD'S FUZZY VARIABLES IN EACH OUTPUT FILE.

FOR A DESCRIPTION OF THE DATA STRUCTURES, CONSTANTS, AND VARIABLES, REFER TO THE DOCUMENTATION AT THE BEGINNING OF PROGRAM LEARNWORD.

*****}

```
PROGRAM OUTSTAT(INPUT,OUTPUT,SPECH,WFILE,SFILE,
                STATSOUT,FUZZYVAR,OPTFUZZY,FZOPT);
```

CONST

```
SPELENGTH=20;
MAXPHONES=20;
MAXPHPLUS1=21;
NOPHONEMES=71;
NOGUESSES=5;
MAXNOWORDS=50;
RCRDLENGTH=40
MAXVECTORS=80;
NOTOAVG=5;
BEGINWORD=1;
NOFUZZYVARS=21;
```

{RCRDLENGTH MUST BE A MULTIPLE OF TWENTY};

TYPE

```
WORDDATA=RECORD
  SPELLING:STRING[SPELLENGTH];
  PHREP:ARRAY[1..MAXPHONES] OF INTEGER;
  WSTHR,WSUBE,WSUBF,WINSE:REAL;
  WINSF,WDELE,WDELF,WDELC:REAL;
  WDCNE,WDCNF,WDCNG:REAL;
  WSFE,WSFF,WCHVE,WCHVF:REAL;
  WSTATE,WSTATF,WSTATG:REAL;
  WTHR1E,WTHR1F,WTHR2E,WTHR2F:REAL;
END (*WORDDATA*);
```

```
WORDSTATS=RECORD
  NOSPOKE,NOATTEMPT,NOCORRECT:INTEGER;
  NDEL:ARRAY[1..MAXPHONES] OF INTEGER;
  XSUBY,XINAPTZ:ARRAY[1..MAXPHONES,1..NOPHONEMES] OF REAL;
  TOTALSCORE:REAL;
  TOTDIFF:REAL;
  MINDIFF:REAL;
END (*WORDSTATS*);
```

STATSFIL=FILE OF WORDSTATS;

```
ISTRING=RECORD
  SPELL:STRING[SPELLENGTH];
  CONTFLAG:INTEGER;
  SPKR:STRING[10];
  ISCORE:ARRAY[1..RCRDLENGTH] OF REAL;
  IPHONE:ARRAY[1..RCRDLENGTH] OF INTEGER;
  IVEERROR:ARRAY[1..RCRDLENGTH] OF STRING[5];
  ICHPHONE:ARRAY[1..NOGUESSES,1..RCRDLENGTH] OF INTEGER;
  ICHVALUE:ARRAY[1..NOGUESSES,1..RCRDLENGTH] OF REAL;
  ISFACTOR:ARRAY[1..RCRDLENGTH] OF REAL
END (*ISTRING*);
```

```
WORDSTRING=FILE OF ISTRING;
WORDFILE= FILE OF WORDDATA;
```

```
FUZZYLIMITS=RECORD
  FUZNAME:ARRAY[1..NOFUZZYVAR] OF STRING[5];
  FUZVAR,MINLIMIT,MAXLIMIT:ARRAY[1..NOFUZZYVAR] OF REAL;
  RECScore:REAL;
END(*FUZZYLIMITS*);
```

FUZOPT=FILE OF FUZZYLIMITS;

VAR

OPTION:INTEGER;

```

WDATA:WORDDATA;
WSTATS:WORDSTATS;
SFILE:STATSFIL;
SPECH:WORDSTRING;
WFILE:WORDFILE;
IOUT:ISTRING;
FZOPT:FUZOPT;
FLIMIT:FUZZYLIMITS;
WORD:STRING[SPELLENGTH];
ENDFLAG:BOOLEAN;
STATSOUT:TEXT;
FUZZYVAR:TEXT;
OPTFUZZY:TEXT;
SPEAKER:STRING[31];
FILENAME:STRING[221];
STHR,SUBE,SUBF:REAL;
INSE,INSF:REAL;
DELE,DELF,DELG:REAL;
DCNE,DCNF,DCNG:REAL;
SFE,SFF:REAL;
CHVE,CHVF:REAL;
STATE,STATF,STATG:REAL;
THR1E,THR1F,THR2E,THR2F:REAL;

```

{*****}

PROCEDURE INITFILE DESCRIPTION

THIS PROCEDURE ACCEPTS THE THREE INITIALS PRECEEDING THE FILES TO BE OPERATED UPON, AND CONCATINATES THEM WITH THE APPROPRIATE FILE NAMES. THE FOLLOWING FILES ARE AFFECTED:

```

WFILE
SFILE
SPECH
FZOPT

```

*****}

PROCEDURE INITFILE;

BEGIN (*INITFILE*)

```
WRITELN;
WRITELN;
WRITELN('ENTER "THREE" INITIALS FOR THE CURRENT *PERMANENT* ');
WRITELN('SPEAKER; OR ENTER "ALL" FOR *NON-PERMANENT* SPEAKERS. ');
WRITE('SPEAKER = ');
READLN(SPEAKER);
WRITELN;
FILENAME:=CONCAT(SPEAKER,'WFILE');
FILETITLE(WFILE,FILENAME);
FILENAME:=CONCAT(SPEAKER,'SFILE');
FILETITLE(SFILE,FILENAME);
FILENAME:=CONCAT(SPEAKER,'SPECH');
FILETITLE(SPECH,FILENAME);
FILENAME:=CONCAT(SPEAKER,'FZOPT');
FILETITLE(FZOPT,FILENAME);
```

END(*PROCEDURE INITFILE*);

{*****}

PROCEDURE GETFUZZYVARS DESCRIPTION

THIS PROCEDURE READS THE VALUES OF THE OVERALL FUZZY VARIABLES
STORED IN THE FILE FUZZYVAR.

*****}

PROCEDURE GETFUZZYVARS;

BEGIN(*GETFUZZYVARS*);

```
RESET(FUZZYVAR);
READLN(FUZZYVAR,STHR,SUBE,SUBF);
READLN(FUZZYVAR,IMSE,INSF);
READLN(FUZZYVAR,DELE,DELF,DELG);
READLN(FUZZYVAR,DCNE,DCNF,DCNG);
READLN(FUZZYVAR,SFE,SFF);
READLN(FUZZYVAR,CHVE,CHVF);
READLN(FUZZYVAR,STATE,STATF,STATG);
READLN(FUZZYVAR,THR1E,THR1F);
```

```

READLN(FUZZYVAR,THR2E,THR2F);

CLOSE(FUZZYVAR);

END(*PROCEDURE GETFUZZYVARS*);

```

```

{*****}

```

``` PROCEDURE GETWORDVARS DESCRIPTION ----- ```

THIS PROCEDURE INITIALIZES THE VALUES OF THE GLOBAL FUZZY VARIABLES TO CORRESPOND TO A WORD'S FUZZY VARIABLES. RECORD WDATA MUST CONTAIN THE APPROPRIATE WORD'S DATA BEFORE THIS PROCEDURE IS CALLED.

```

*****}

```

```

PROCEDURE GETWORDVARS;

BEGIN(*GETWORDVARS*)

```

```

    WITH WDATA DO
    BEGIN(*WITH*)
        STHR:=WSTHR;
        SUBE:=WSUBE;
        SUBF:=WSUBF;
        INSE:=WINSE;
        INSF:=WINSF;
        DELE:=WDELE;
        DELF:=WDELF;
        DELG:=WDELG;
        DCNE:=WDCNE;
        DCNF:=WDCNF;
        DCNG:=WDCNG;
        SFE:=WSFE;
        SFF:=WSFF;
        CHVE:=WCHVE;
        CHVF:=WCHVF;
        STATE:=WSTATE;
    END(*WITH*);

```

```

STATF:=WSTATF;
STATG:=WSTATG;
THR1E:=WTHR1E;
THR1F:=WTHR1F;
THR2E:=WTHR2E;
THR2F:=WTHR2F;

```

```

END(*WITH*);

```

```

END(*PROCEDURE GETWORDVARS*);

```

```

{*****}

```

PROCEDURE AUTOOUT DESCRIPTION

THIS PROCEDURE OUTPUTS THE INFORMATION STORED IN SFILE AND WFILE, AND THE VALUES OF THE OVERALL AND WORD FUZZY VARIABLES FOR THE WORD SPECIFIED BY THE VARIABLE WORDNO.

```

*****}

```

```

PROCEDURE AUTOOUT(WORDNO:INTEGER);

```

```

VAR

```

```

    AVESCORE:REAL;
    I,J:INTEGER;

```

```

BEGIN(*AUTOOUT*)

```

```

    SEEK(WFILE,WORDNO);
    GET(WFILE);
    WDATA:=WFILE^;
    WRITELN(STATSOUT);
    WRITELN(STATSOUT,'THE STATISTICS FOR ',WDATA.SPELLING,' FOLLOW.');
```

```

WRITELN(STATSOUT,'# OF TIMES SPOKEN = ',NOSPOKE);
WRITELN(STATSOUT,'# OF RECOGNITION ATTEMPTS = ',NOATTEMPT);
WRITELN(STATSOUT,'# OF TIMES CORRECTLY RECOGNIZED = ',NOCORRECT);
IF NOATTEMPT(>0) THEN
BEGIN (*IF*)
    AVESCORE:=TOTALSCORE/NOATTEMPT;
    WRITELN(STATSOUT,'AVERAGE RECOGNITION SCORE = ',AVESCORE:8:3);
END (*IF*)
ELSE
BEGIN (*ELSE*)
    AVESCORE:=0.0;
    WRITELN(STATSOUT,'AVERAGE RECOGNITION SCORE = ',AVESCORE:4);
END (*ELSE*);
WRITELN(STATSOUT);
IF NOCORRECT(>0) THEN
BEGIN (*IF*)
    AVESCORE:=TOTDIFF/NOCORRECT;
    WRITELN(STATSOUT,'AVERAGE DIFFERENCE BETWEEN WORD SCORE');
    WRITELN(STATSOUT,'AND NEXT HIGHEST SCORE = ',AVESCORE:8:3);
END (*IF*)
ELSE
BEGIN (*ELSE*)
    AVESCORE:=0.0;
    WRITELN(STATSOUT,'AVERAGE DIFFERENCE BETWEEN WORD SCORE');
    WRITELN(STATSOUT,'AND NEXT HIGHEST SCORE = ',AVESCORE:4:2);
END (*ELSE*);
WRITELN(STATSOUT);
WRITELN(STATSOUT,'MINIMUM DIFFERENCE BETWEEN WORD SCORE');
WRITELN(STATSOUT,'AND NEXT HIGHEST SCORE = ',MINDIFF);
WRITELN(STATSOUT);
GETFUZZYVARS;
WRITELN(STATSOUT);
WRITELN(STATSOUT);
WRITELN(STATSOUT,'THE OVERALL FUZZY VARIABLES THAT WERE USED FOLLOW');
WRITELN(STATSOUT);
WRITELN(STATSOUT,'STHR = ',STHR,' SUBE = ',SUBE,' SUBF = ',SUBF);
WRITELN(STATSOUT,'INSE = ',INSE,' INSF = ',INSF);
WRITELN(STATSOUT,'DELE = ',DELE,' DELF = ',DELF,' DELG = ',DELG);
WRITELN(STATSOUT,'DCNE = ',DCNE,' DCNF = ',DCNF,' DCNG = ',DCNG);
WRITELN(STATSOUT,'SFE = ',SFE,' SFF = ',SFF);
WRITELN(STATSOUT,'CHVE = ',CHVE,' CHVF = ',CHVF);
WRITELN(STATSOUT,'STATE = ',STATE,' STATF = ',STATF,' STATG = ',STATG);
WRITELN(STATSOUT,'THR1E = ',THR1E,' THR1F = ',THR1F);
WRITELN(STATSOUT,'THR2E = ',THR2E,' THR2F = ',THR2F);
WRITELN(STATSOUT);
WRITELN(STATSOUT);
GETWORDVARS;
WRITELN(STATSOUT);
WRITELN(STATSOUT);
WRITELN(STATSOUT,'THE WORD FUZZY VARIABLES FOLLOW');
WRITELN(STATSOUT);
WRITELN(STATSOUT,'WSTHR = ',STHR,' WSURE = ',SUBE,' WSURF = ',SUBF);
WRITELN(STATSOUT,'WINSE = ',INSE,' WINSF = ',INSF);
WRITELN(STATSOUT,'WDELE = ',DELE,' WDELF = ',DELF,' WDELG = ',DELG);

```

```

WRITELN(STATSOUT,'WDCNE = ',DCNE,' WDCNF = ',DCNF,' WDCNG = ',DCNG);
WRITELN(STATSOUT,'WSFE = ',SFE,' WSFF = ',SFF);
WRITELN(STATSOUT,'WCHVE = ',CHVE,' WCHVF = ',CHVF);
WRITELN(STATSOUT,'WSTATE= ',STATE,' WSTATF= ',STATF,' WSTATG= ',STATG);
WRITELN(STATSOUT,'WTHR1E= ',THR1E,' WTHR1F= ',THR1F);
WRITELN(STATSOUT,'WTHR2E= ',THR2E,' WTHR2F= ',THR2F);
WRITELN(STATSOUT);
WRITELN(STATSOUT);
WRITELN(STATSOUT);
WRITELN(STATSOUT,'# TIMES EACH WORD PHONEME IS DELETED');
WRITELN(STATSOUT,'-----');
WRITE(STATSOUT,'PHONEME REP = ');
FOR I:= 1 TO 10 DO
BEGIN (*FOR*)
    WRITE(STATSOUT,WDATA.PHREPC[1]:5);
END(*FOR*);
WRITELN(STATSOUT);
WRITE(STATSOUT,'# DELETED = ');
FOR I:=1 TO 10 DO
BEGIN (*FOR*)
    WRITE(STATSOUT,NDEL[1]:5);
END (*FOR*);
WRITELN(STATSOUT);
WRITELN(STATSOUT);
WRITE(STATSOUT,'PHONEME REP = ');
FOR I:= 11 TO 20 DO
BEGIN (*FOR*)
    WRITE(STATSOUT,WDATA.PHREPC[1]:5);
END(*FOR*);
WRITELN(STATSOUT);
WRITE(STATSOUT,'# DELETED = ');
FOR I:=11 TO 20 DO
BEGIN (*FOR*)
    WRITE(STATSOUT,NDEL[1]:5);
END (*FOR*);
WRITELN(STATSOUT);
WRITELN(STATSOUT);
WRITELN(STATSOUT);
WRITELN(STATSOUT);
WRITELN(STATSOUT);
WRITELN(STATSOUT,'# TIMES PHONEME X SUBSTITUTED FOR WORD PHONE');
WRITELN(STATSOUT,'-----');
WRITELN(STATSOUT);
WRITELN(STATSOUT,' ':40,'WORD PHONEMES');
WRITE(STATSOUT,' ':4,'X',' ':5);
FOR I:=1 TO 10 DO
BEGIN (*FOR*)
    WRITE(STATSOUT,WDATA.PHREPL[1]:8);
END (*FOR*);
WRITELN(STATSOUT);
WRITELN(STATSOUT,'.':60);
WRITELN(STATSOUT);
FOR J:= 1 TO NOPHONEMES DO
BEGIN (*FOR*)
    WRITE(STATSOUT,J:5,' ':5);

```

```

    FOR I:=1 TO 10 DO
    BEGIN (*FOR*)
        WRITE(STATSOUT,XSUBY[I,J]:8:3);
    END (*FOR*);
    WRITELN(STATSOUT);
END (*FOR*);
WRITELN(STATSOUT);
WRITELN(STATSOUT);
IF WDATA.PHREP[11]<>00 THEN
BEGIN (*IF*)
    WRITELN(STATSOUT,' ':40,'WORD PHONEMES');
    WRITE(STATSOUT,' ':4,'X',' ':5);
    FOR I:=11 TO 20 DO
    BEGIN (*FOR*)
        WRITE(STATSOUT,WDATA.PHREP[I]:8);
    END (*FOR*);
    WRITELN(STATSOUT);
    WRITELN(STATSOUT,' ':60);
    WRITELN(STATSOUT);
    FOR J:= 1 TO NOPHONEMES DO
    BEGIN (*FOR*)
        WRITE(STATSOUT,J:5,' ':5);
        FOR I:=11 TO 20 DO
        BEGIN (*FOR*)
            WRITE(STATSOUT,XSUBY[I,J]:8:3);
        END (*FOR*);
        WRITELN(STATSOUT);
    END (*FOR*);
END(*IF*);
    WRITELN(STATSOUT);
    WRITELN(STATSOUT);
    WRITELN(STATSOUT,'* TIMES PHONEME X INSERTED FOR WORD PHONE');
    WRITELN(STATSOUT,'-----');
    WRITELN(STATSOUT);
    WRITELN(STATSOUT,' ':40,'WORD PHONEMES');
    WRITE(STATSOUT,' ':4,'X',' ':5);
    FOR I:=1 TO 10 DO
    BEGIN (*FOR*)
        WRITE(STATSOUT,WDATA.PHREP[I]:8);
    END (*FOR*);
    WRITELN(STATSOUT);
    WRITELN(STATSOUT,' ':60);
    WRITELN(STATSOUT);
    FOR J:= 1 TO NOPHONEMES DO
    BEGIN (*FOR*)
        WRITE(STATSOUT,J:5,' ':5);
        FOR I:=1 TO 10 DO
        BEGIN (*FOR*)
            WRITE(STATSOUT,XINAFIZ[I,J]:8:3);
        END (*FOR*);
        WRITELN(STATSOUT);
    END (*FOR*);
    WRITELN(STATSOUT);
    WRITELN(STATSOUT);

```

```

IF WDATA.PHREP[11]<>00 THEN
BEGIN(*IF*)
  WRITELN(STATSOUT,' ':40,'WORD PHONEMES');
  WRITE(STATSOUT,' ':4,'X',' ':5);
  FOR I:=11 TO 20 DO
  BEGIN (*FOR*)
    WRITE(STATSOUT,WDATA.PHREP[I]:8);
  END (*FOR*);
  WRITELN(STATSOUT);
  WRITELN(STATSOUT,' ':60);
  WRITELN(STATSOUT);
  FOR J:= 1 TO NOPHONEMES DO
  BEGIN (*FOR*)
    WRITE(STATSOUT,J:5,' ':5);
    FOR I:=11 TO 20 DO
    BEGIN (*FOR*)
      WRITE(STATSOUT,X1NAFTZ[I,J]:8:3);
    END (*FOR*);
    WRITELN(STATSOUT);
  END (*FOR*);
END (*IF*);

  WRITELN('THE STATISTICS FOR ',WDATA.SPELLING,' ARE STORED');
  WRITELN('IN FILE "',FILENAME,'"');

END(*WITH*);

CLOSE(STATSOUT);

END(*PROCEDURE AUTOOUT*);

{*****}

PROCEDURE OUTSTATS DESCRIPTION
-----

  THIS PROCEDURE ACCEPTS THE NAME OF THE WORD WHOSE INFORMATION
  IS TO BE OUTPUT AND THE NAME OF THE OUTPUT FILE, AND THEN CALLS
  PROCEDURE AUTOOUT.

{*****}

```

PROCEDURE OUTSTATS;

VAR

I,J:INTEGER;
WORD:STRING[SPELLENGTH];
TEMPWORD:STRING[SPELLENGTH];
WORDNO:INTEGER;
AVESCORE:REAL;
ANSWER:STRING[3];

BEGIN (*OUTSTATS*)

INITFILE;
RESET(WFILE);
WRITELN('INPUT NAME OF OUTPUT FILE');
WRITE('FILENAME = ');
READLN(FILENAME);
WRITELN;
FILETITLE(STATSOUT,FILENAME);
REWRITE(STATSOUT);
RESET(SFILE);
WORDNO:=0;
SEEK(WFILE,WORDNO);
GET(WFILE);
WRITELN;
WRITELN('THE FOLLOWING IS THE CURRENT VOCABULARY');
WRITELN;
WHILE NOT EOF(WFILE) DO
BEGIN (*WHILE*)
WDATA:=WFILE^;
WRITELN(WDATA.SPELLING);
GET(WFILE);
END (*WHILE*);
WRITELN;
WRITELN;
WRITELN('INPUT WORD TO OUTPUT STATISTICS');
WRITELN;
WRITE('WORD = ');
READLN(WORD);
WRITELN;
FOR I:=(LENGTH(WORD)) TO (SPELLENGTH-1) DO
BEGIN(*FOR*)
WORD:=CONCAT(WORD,' ');
END(*FOR*);
RESET(WFILE);
TEMPWORD:='';
WORDNO:=0;
WHILE (NOT EOF(WFILE))AND(WORD<>TEMPWORD) DO
BEGIN (*WHILE*)
SEEK(WFILE,WORDNO);
GET(WFILE);

```

        WDATA:=WFILE^;
        GET(WFILE);
        TEMPWORD:=WDATA.SPELLING;
        WORDNO:=WORDNO+ 1
        END (*WHILE*);
    IF (WORD<>TEMPWORD) THEN
        BEGIN (*IF*)
            WRITELN;
            WRITELN('WORD DOES NOT EXIST.');
```

END (*IF*)

```

        ELSE
            BEGIN(*ELSE*)
                WORDNO:=WORDNO-1;
                AUTOOUT(WORDNO);

            END(*ELSE*);

    END (*PROCEDURE OUTSTATS*);
```

```

{*****}
```

PROCEDURE ALLOUT DESCRIPTION

THIS PROCEDURE OUTPUTS THE INFORMATION FOR EVERY WORD IN THE CURRENT VOCABULARY TO FILES WHOSE NAMES ARE GIVEN BY THE CONCATENATION OF THE CHARACTER "S" AND THE SPELLING OF THE WORD.

```

{*****}
```

```
PROCEDURE ALLOUT;
```

```
VAR
```

```
    WORDNO:INTEGER;
```

```
BEGIN(*ALLOUT*)
```

```
    INITFILE;
```

```
    RESET(SFILE);
```

```
    WORDNO:=0;
```

```
    RESET(WFILE);
```

```
    WHILE NOT EOF(WFILE) DO
```

```

BEGIN(*WHILE*)
    SEEK(WFILE,WORDNO);
    GET(WFILE);
    WDATA:=WFILE^;
    FILENAME:=CONCAT('S',WDATA.SPELLING);
    FILETITLE(STATSOUT,FILENAME);
    REWRITE(STATSOUT);
    AUTOOUT(WORDNO);
    SEEK(WFILE,WORDNO);
    GET(WFILE);
    GET(WFILE);
    WORDNO:=WORDNO+1;
END(*WHILE*);

END(*PROCEDURE ALLOUT*);

BEGIN(*PROGRAM OUTSTAT*)

    WRITELN;
    WRITELN('CHOOSE OPTION FROM FOLLOWING LIST');
    WRITELN;
    WRITELN('1 - OUTPUT STATISTICS FOR A GIVEN WORD.'):
    WRITELN('2 - OUTPUT STATISTICS FOR ALL WORDS.'):
    WRITELN;
    WRITE('OPTION = ');
    READLN(OPTION);
    WRITELN;

    IF OPTION = 1 THEN
        BEGIN(*IF*)
            OUTSTATS;
        END(*IF*)
    ELSE IF OPTION = 2 THEN
        BEGIN(*ELSE*)
            ALLOUT;
        END(*ELSE*)
    ELSE
        BEGIN(*ELSE*)
            WRITELN('OPTION DOES NOT EXIST');
        END(*ELSE*);

END (*PROGRAM OUTSTAT*).

```

```

C *****
C *****
C **
C **      PROGRAM:  P H D I S T      **
C **
C **      WRITTEN BY:LT KARL SEELANDT      **
C **
C **      DATE:19 NOV 1981      **
C **
C **
C **      REVISED BY: LT JERRY MONTGOMERY      **
C **
C **      DATE: 17 APRIL 1982      **
C **
C *****
C *****
C
C NOTE: THIS ROUTINE IS FOR FORTRAN 5
C
C
C*****  Tis program is a modification of a program
C written by Lt. Karl Seelandt here at AFIT. The purpose
C of this program is to determine the distances between
C spectrographic frequency components of an input
C speech sentence and a set of prototype phoneme
C templates. The prototypes are stored
C in DISK3 and the sentence is read from DSPEECH.
C The routine must be set up to manipulate
C the digital speech in the same exact manner
C that the prototype phonemes were formed since it
C creates the spectrogram of the speech before
C actually doing the distance measurement.
C
C*****  This program is set up to do any of the
C manipulations that the program SGRAM2 can do.
C This allows the study of various manipulations
C and their effect on the recognition accuracy.
C
C
C INITIALIZE ARRAYS
C
C   DIMENSION ISENT(512,32),IPHONE(16,32),RESULT(512),B(64),IBI(64)
C   DIMENSION IDSP(512),RMINMAX(512,2)
C   COMPLEX DTARRAY(128)
C   INTEGER DSPEECH(13)
C
C INITIALIZE CONTROL VARIABLES FOR STANDARD OPERATION.
C
C   IPRE=1 ;FLAG TO DO PREEMPHASIS
C   IDB=6 ;NUMBER OF DB'S PER OCTAVE PREEMPHASIS
C   FREQ=500.0;STARTING FREQUENCY FOR PREEMPHASIS
C   IPRESS=2 ;FLAG TO NOT DO FREQUENCY COMPONENT COMPRESSION
C   ISIZE=64 ;NUMBER OF SAMPLES PER DFT

```

```

ILENGTH=5;LENGTH OF PHONEMES IN TIME INCREMENTS
NUMPP=39 ;NUMBER OF PROTOTYPE PHONEMES
SFREQ=8000.0 ;SAMPLING FREQUENCY
;INITIALIZE MIN-MAX VARIABLES USED TO KEEP TRACK OF THE MINIMUM
;AND MAXIMUM DISTANCES DETERMINED FOR EACH TIME FRAME.
DO 500 I=1,512
  RMINMAX(I,1)=100000000.0 ;MAX POSSIBLE DISTANCE FOR MINIMUM VARIABLE
  RMINMAX(I,2)=0.0 ;MIN POSSIBLE DISTANCE FOR MAXIMUM VARIABLE.
500   CONTINUE
C
C INPUT CONTROL VARIABLES AND OPEN FILES.
C
  ACCEPT"ENTER, name of speechfile to be recognized:"
  READ(11,510)DSPEECH(1)
510   FORMAT(S13)
C
  OPEN 3,"RESTEMP" ;
  CALL OPEN(5,DSPEECH,1,512,IER1) ;speechfile data
  CALL CHECK (IER1)
  ACCEPT "FIRST BLOCK TO READ=",ISTART
  ACCEPT "LAST BLOCK TO READ=",ILAST
  ISLEN=((ILAST+1)*256)/ISIZE ;LENGTH OF SENTENCE(VECTORS)
  ISIZEH=ISIZE/2 ;NUMBER OF COMPONENTS PER VECTOR
  ISAMP=((256.0/ISIZE)*ISTART)+1 ;NUMBER OF 1ST TIME INCREMENT USED.
  ISAMP1=ISAMP ;SAVE THIS INITIAL VALUE FOR LATER
  DELETE "DCOMPS"
  OPEN 4,"DCOMPS",LEN=ISIZE
C
  JSIZE=ISIZE*ILENGTH+2 ;RECORD SIZE OF PHONEME FILE
C
C
  IHAM=1
  NUMPP=71
  ENTHRES=60
C
C
  OPEN 2,"DISK3HAM",LEN=JSIZE
C
C
  CALL FGTIME(IHOUR,IMIN,ISEC) ;GET STARTING TIME
  ;
  ;
  ;
  ; SECTION OF PROGRAM TO FORM SENTENCE ARRAY.
  ;
  ;
  DO 1000 IM=ISTART,ILAST
  CALL RDBLK(5,IM,IDSP,1,IER) ;READ IN DIGITIZED SPEECH
  IF(IER.NE.1) TYPE "ERROR ON RDBLK,IER=",IER
  J1=0 ;OFFSET USED TO SPLIT UP BLOCK OF SPEECH DATA
600   CONTINUE
  ;SET UP COMPLEX ARRAY FOR DFT
  ;And iff requested use aHamming window on the

```

```

        ;samples as shown in Oppenheim and Shafer's
        ;Digital Signal Processing (p. 242).
        ;
DO 610 J=1,ISIZE
B(J)=IDSP(J+J1)
C
WINDO = 0.54 -0.46 * COS( 2.0 *
/ 3.1415926536 * (J-1)/ISIZE )
C
B(J) = WINDO * B(J)          ;window dspeech points
C
DTARAY(J)=B(J)
610    CONTINUE
LARAY=ISIZE ;# OF POINTS FOR DFT
INVER=0 ;SPECIFIES A FORWARD TRANSFORM
CALL DFT5(DTARAY,LARAY,INVER)
;Must be sure that the magnitudes of the
;two differently windowed components are
;are equivalent, do this be increasing
;magnitude of Hamming windowed components
;assuming magnitude of Rect. windowed
;components = 1.
;
;FIND MAGNITUDE OF FREQUENCY COMPONENTS
DO 625 I=1,ISIZEH
B(I)=CABS(DTARAY(I)/LARAY)
C
IF (IHAM.EQ.1) B(I) = B(I) / 0.54
C
625    CONTINUE
C
C PREEMPHASIZE HIGH FREQUENCY COMPONENTS.
C
IF (IPRE.NE.1) GO TO 45
;FIND 1ST COMPONENT TO START PREEMPHASIS AT
IFREQ1=IFIX(FREQ/SFREQ*ISIZE)+1
DO 700 I=IFREQ1,ISIZEH
;CONVERT INTEGER VARIABLES TO REALS
RI=I
RIFREQ1=IFREQ1
RIDB=IDB
;MODIFY MAGNITUDE OF COMPONENTS
B(I)=B(I)*(10**((RIDB/20*ALOG10(RI/RIFREQ1)/ALOG10(2.0)))
700    CONTINUE
C
C FREQUENCY VECTOR COMPRESSED TO 16 CHANNELS IF COMPRESSION WAS SELECTED.
C
45 IF (IPRESS.NE.1) GO TO 30
;FIND NUMBER OF COMPONENTS TO COMBINE INTO 1 COMPONENT
JNUM=ISIZEH/16
JJ1=1
JJ2=JNUM
DO 20 I=1,16
SUM=0.0

```

```

      DO 10 J=JJ1,JJ2
10  SUM=SUM+B(J)
      B(I)=SUM
      JJ1=JJ1+JNUM
      JJ2=JJ2+JNUM
20  CONTINUE
C
C  NORMALIZATION OF ENERGY IN SIGNAL
C
30  IF (IPRESS.EQ.1) ISIZEH=16
      SUME=0.0
      DO 33 J=1,ISIZEH
      SUME=SUME+(B(J)**2)
33  CONTINUE
      ENERGY=SQRT(SUME)
      IF (ENERGY.GT.ENTHRES) GO TO 32
      ;INCREASE VALUE OF ENERGY TO ATTENUATE COMPONENTS
      ENERGY=ENTHRES*5
32  CONTINUE
      IMULT=32760 ;RANGE OF COMPONENTS AFTER NORMALIZATION.
      DO 34 J=1,ISIZEH
      B(J)=(B(J)/ENERGY)*IMULT
34  CONTINUE
C
C  ARRAY VALUES CONVERTED TO INTEGER FORM
C
      DO 240 JJ=1,ISIZEH
      IBI(JJ)=IFIX(B(JJ))+1
240  CONTINUE
      ;SAVE COMPONENTS INCASE NEEDED LATER BY 'PPGEN'
      IREC=ISAMP ;CORRECT RECORD TO WRITE TO IN 'DCOMPS'
      CALL WRITR(4,IREC,IBI,1,IER21)
      CALL CHECK (IER21)
C
C  FORM SENTENCE ARRAY
C
      DO 215 I=1,ISIZEH
      ISENT(ISAMP,I)=IBI(I)
215  CONTINUE
      ISIZEH=ISIZE/2 ;RESTORE VALUE IN CASE ALTERED.
      ISAMP=ISAMP+1
      J1=J1+ISIZE
      IF(J1.LT.256) GOTO 600 ;LOOP IF MORE VECTORS POSSIBLE FROM BLOCK
1000 CONTINUE
      CLOSE 4 ;Close the DCOMPS file.
C
C  READ IN PROTOTYPE PHONEMES AND DETERMINE RECOGNITION RESULTS.
C
      IF(IPRESS.EQ.1) ISIZEH=16
      ;LOOP TO SELECT 1 PROTOTYPE AT A TIME
      DO 900 IPP=1,NUMPP
      IREC=IPP ;RECORD TO READ FROM DISK3
      CALL READRW(2,IREC,IDSF,1,IER7)
      IF(IER7.NE.1)TYPE"ERROR IER7 =",IER7

```

```

CALL CHECK (IER7)
JOFFSET=0 ;OFFSET USED TO SETUP THE ARRAY IPHONE
DO 920 I=1,ILENGTH
DO 910 J=1,ISIZEH
IPHONE(I,J)=IDSP(J+JOFFSET+1) ;ADD 1 TO SKIP VARIABLE FOR NUMBER OF
                                ;TIMES THE PHONEME HAS BEEN MODIFIED.
910      CONTINUE
920      JOFFSET=JOFFSET+ISIZEH
C
C DETERMINE DISTANCE.
C
      RMAX=0.0
      INOSHFT=ISAMP1-1 ;MIN SHIFT REQUIRED TO GET ALL DISTANCES
      IMAXSHFT=ISLEN-ILENGTH-1 ;MAX SHIFT REQUIRED TO GET ALL DISTANCES.
      ;LOOP THAT GETS ALL DISTANCES FOR THIS PROTOTYPE
      DO 880 ISHIFT=INOSHFT,IMAXSHFT
      RSUMDIS=0.0 ;USED TO SUM UP DISTANCE
      ;CALCULATE 2-DIMENSIONAL DISTANCE FOR THIS TIME FRAME
      DO 850 I=1,ILENGTH
      IS=I+ISHIFT ;AMOUNT TO SHIFT
      DO 800 J=1,ISIZEH
800      RSUMDIS=RSUMDIS+IABS(ISENT(IS,J)-IPHONE(I,J))
850      CONTINUE
      ;TEST FOR MAX-MIN DISTANCES FOR THIS TIME FRAME
      IF(RSUMDIS.GT.RMAX) RMAX=RSUMDIS
      IF(RSUMDIS.LT.RMINMAX(ISHIFT+1,1)) RMINMAX(ISHIFT+1,1)=RSUMDIS
      IF(RSUMDIS.GT.RMINMAX(ISHIFT+1,2)) RMINMAX(ISHIFT+1,2)=RSUMDIS
      ;SAVE RESULT FOR THIS TIME FRAME
      RESULT(ISHIFT+1)=RSUMDIS
C
C FIND DISTANCE BETWEEN EACH VECTOR OF EACH SECTION OF SPEECH
C TO CHECK FOR POSSIBLE PHONEME-PHONEME TRANSITIONS. DO ONLY ON 1ST
C PASS THRU THIS SECTION OF PROGRAM.
C
X      IF(IPP.NE.1)GOTO 880 ;SKIP THIS IF ALREADY DONE ON 1ST PASS THRU
X      TRDIFF=0.0 ;USED TO SUM UP DIFFERENCE
X      DO 870 J=1,ISIZEH
X870      TRDIFF=TRDIFF+IABS(ISENT(ISHIFT+1,J)-ISENT(ISHIFT+2,J))
X      CALL FSEEK(1,ISHIFT) ;SELECT CORRECT RECORD
X      WRITE BINARY(1) TRDIFF ;SAVE TRANSITION DISTANCE
880      CONTINUE
      ;
      ;FILL UP BEGINNING OF TRANSITION DISTANCE FILE WITH A LARGE VALUE
      ;
X      TRDIFF=1000000000.0
X      DO 885 I=0,INOSHFT
X      CALL FSEEK(1,I)
X885      WRITE BINARY(1) TRDIFF
C
C FILL UP REMAINDER OF RESULT ARRAY W/ MAX DISTANCE
C
      IINC=IMAXSHFT+1
      DO 890 I=IINC,ISLEN
890      RESULT(I)=RMAX

```

```

C
C WRITE RESULTS OUT TO TEMPORARY FILE.
C
  DO 895 I=ISAMP1,ISLEN
895    WRITE BINARY(3)RESULT(I)
      TYPE "PHONEMES COMPLETED= <7><7><7>",IPP
900    CONTINUE    ;END OF LOOP TO CALCULATE DISTANCE FOR ALL PHONEMES.
C
C RECONFIGURE RESULTS TO BE COMPATIBLE WITH INTPLOT AND OUTPUT TO "CORRAR".
C
C
X   CLOSE 1                      ;transtion file closed.
C
C*****    Store a scale factor for each time frame.  This
C   factor will give a measure of how accurate a choice

C   was made in recognizing phonemes for this time period.
C   This scale will be used by the CHOICES5 routine.
C
C
  DELETE "SFACTOR"
  OPEN 1,"SFACTOR"                      ;scale factor
    RMAXMAX=0.0
    DO 993 ISC=ISAMP1,ISLEN

      IF (RMINMAX(ISC,1).GT.900000000.00)GOTO 993
      IF (RMINMAX(ISC,1).GT.RMAXMAX) RMAXMAX=RMINMAX(ISC,1)
993    CONTINUE
      DO 990 ISC=ISAMP1,ISLEN              ;
        RMINMIN=RMINMAX(ISC,1)/RMAXMAX
        WRITE (1,693) RMINMIN              ;store 'em.
990    CONTINUE                          ;
      IZERO = 0.0                          ;fill end of
      DO 991 JZ=1,4                        ;file with 0.
        WRITE BINARY(1) IZERO              ;
991    CONTINUE                          ;
      CLOSE 1 ;keep channel count as low as possible
          ;or the program will DIE.
C
C
  DELETE "RFACTOR"                      ;
  OPEN 1,"RFACTOR"                      ;range factor
C
  REWIND 3
  DELETE "CORRAR"
  CALL CFILW("CORRAR",2,IER99)
  IF(IER99.NE.1)TYPE"ERROR on CFILW IER99= ",IER99
C
C
  CALL OPEN(4,"CORRAR",2,512,IER18)

```

```

IF(IER18.NE.1)TYPE"ERROR IER18=",IER18
C
ISTBLK=0
;
;LOOP TO OUTPUT 2 RECORDS FOR EACH PHONEME
;
DO 2000 IM=1,NUMPP
;USE MAX-MIN DISTANCES TO SCALE RESULTS FOR EACH TIME FRAME.
;0 IS ASSIGNED TO PHONEME W/MAX DISTANCE.
;32767(MAX INTEGER SIZE) IS ASSIGNED TO PHONEME W/MIN DISTANCE.
DO 1500 I=ISAMP1,ISLEN
RFACTOR=RMINMAX(I,2)-RMINMAX(I,1)
C
C***** Store the weighting factor in a file for use by
C CHOICE5. This statistic will give a measure of how
C much of a variance there is between the best and the
C worst phoneme match for each time period.
C
C
IF(IM.GT.1)GO TO 1300 ;do only on first
;pass thru.
WRITE (1,696) RFACTOR
693 FORMAT(F11.8)
696 FORMAT(F11.1) ;factor format stmt.
1300 CONTINUE
C
C
READ BINARY(3) RESULT(I)
IDSP(I)=IFIX((RMINMAX(I,2)-RESULT(I))/RFACTOR*32767)
1500 CONTINUE
;
;ZERO FILL REMAINDER OF RESULT FILE
;
;FILL BEGINNING OF FILE FOR UNUSED TIME FRAMES.
DO 1700 I=1,INOSHFT
IDSP(I)=0
1700 CONTINUE
;FILL END OF FILE FOR UNUSED TIME FRAMES.
IINC=ISLEN+1
DO 1800 I=IINC,512
IDSP(I)=0
1800 CONTINUE
;OUTPUT RESULTS.
CALL WRBLK(4,ISTBLK,IDSP,2,IER20)
IF(IER20.NE.1)TYPE"ERROR IER20=",IER20
CALL CHECK(IER20)
ISTBLK=ISTBLK+2
2000 CONTINUE
C
C CLOSE FILES AND TERMINATE PROGRAM.
C
CALL FCTIME(IHOUR1,IMIN1,ISEC1)
WRITE (10,99)IHOUR,IMIN,ISEC
C

```

```

WRITE (10,98)IHOUR1,IMIN1,ISEC1
C
99 FORMAT(" ", "STARTING TIME = ",3I3)
98 FORMAT(" ", "FINISHED TIME = ",3I3)
C
TYPE "{15}"
TYPE "SENTENCE LENGTH IS = ",ISLEN
TYPE "NUMBER OF PHONEMES = ",NUMPP
C
CALL RESET
DELETE "RESTEMP1"
WRITE (10) "<7><7><7><7><7>"
STOP
END

```

```

C      THIS PROGRAM TAKES UP TO
C      UP TO 75 PHONEMES.....
C
C      . . . . .
C      *****
C      *****
C      **
C      **      PROGRAM:  C H O I C E 5      **
C      **
C      **      WRITTEN BY: LT KARL SEELANDT      **
C      **
C      **      DATE: 1 DEC 1981      **
C      **
C      **      REVISED BY: JERRY MONTGOMERY      **
C      **
C      **      DATE: 15 APRIL      **
C      *****
C      *****
C
C
C
C***** N O T E : THIS ROUTINE IS FOR FORTRAN 5 ONLY!!!!
C
C
C***** This program is used with the program PHDIST
C to locate and tabulate useful information about
C sound units in speech. The routine is currently
C set up to handle a maximum of 71 phonemes.
C The files CORRAR, RFACTOR and SFACTOR are
C created by the program PHDIST and contain the
C distances from each prototype sound unit template
C to each section of speech, along with scaling and
C range factors used in determining these distances.
C
C***** There are two output files generated by this
C program, OUT1 and OUT2. The first contains a
C tabular listing of the top five phoneme choices
C for each time period, along with the scale factor,
C The second file (OUT2) is identical to OUT1 except
C that the noise occurring before and after the actual
C speech has been removed.
C
C
C
C

```

```

      INTEGER SYMBOL(71)
      DIMENSION ISAID(50),ISPKR(10),IDATE(3)
      DIMENSION IBLK(512),ISTORE(200,71)
      DIMENSION ICH1VAL(200),ICH2VAL(200),ICH3VAL(200)
      DIMENSION ICH4VAL(200),ICH5VAL(200),ICH1PHO(200),ICH2PHO(200)
      DIMENSION ICH3PHO(200),ICH4PHO(200),ICH5PHO(200)
      COMMON / NAME / SYMBOL

```

AD-A124 851

ISOLATED WORD RECOGNITION USING FUZZY SET THEORY(U) AIR 4/4
FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING G J MONTGOMERY DEC 82 AFIT/GE/EE/82D-74

UNCLASSIFIED

F/G 5/8

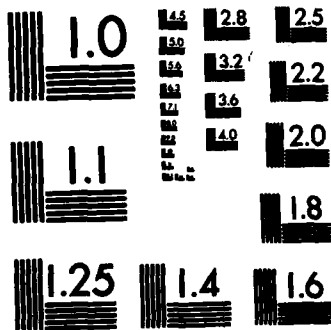
NL



END

FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```

C      THIS PROGRAM TAKES UP TO
C      UP TO 75 PHONEMES.....
C
C      . . . . .
C      *****
C      *****
C      **
C      **      PROGRAM:  C H O I C E 5      **
C      **
C      **      WRITTEN BY: LT KARL SEELANDT      **
C      **
C      **      DATE: 1 DEC 1981      **
C      **
C      **      REVISED BY: JERRY MONTGOMERY      **
C      **
C      **      DATE: 15 APRIL      **
C      *****
C      *****
C
C
C
C***** N O T E : THIS ROUTINE IS FOR FORTRAN 5 ONLY!!!!
C
C
C***** This program is used with the program PHDIST
C to locate and tabulate useful information about
C sound units in speech. The routine is currently
C set up to handle a maximum of 71 phonemes.
C The files CORRAR, RFACTOR and SFACOR are
C created by the program PHDIST and contain the
C distances from each prototype sound unit template
C to each section of speech, along with scaling and
C range factors used in determining these distances.
C
C***** There are two output files generated by this
C program, OUT1 and OUT2. The first contains a
C tabular listing of the top five phoneme choices
C for each time period, along with the scale factor,
C The second file (OUT2) is identical to OUT1 except
C that the noise occurring before and after the actual
C speech has been removed.
C
C
C
C

```

```

      INTEGER SYMBOL(71)
      DIMENSION ISAID(50),ISPKR(10),IDATE(3)
      DIMENSION IBLK(512),ISTORE(200,71)
      DIMENSION ICH1VAL(200),ICH2VAL(200),ICH3VAL(200)
      DIMENSION ICH4VAL(200),ICH5VAL(200),ICH1PHO(200),ICH2PHO(200)
      DIMENSION ICH3PHO(200),ICH4PHO(200),ICH5PHO(200)
      COMMON / NAME / SYMBOL

```

C
C
C

```
DATA SYMBOL/"01","02","03","04","05","06","07",  
+ "08","09","10","11","12","13","14","15",  
+ "16","17","18","19","20","21","22","23",  
+ "24","25","26","27","28","29","30","31",  
+ "32","33","34","35","36","37","38","39",  
+ "40","41","42","43","44","45","46","47",  
+ "48","49","50","51","52","53","54","55",  
+ "56","57","58","59","60","61","62","63",  
+ "64","65","66","67","68","69","70","71"/
```

C
C
C
C

OPEN FILES,PRINT HEADING, INPUT CONTROL VARIABLES.

C

```
CALL DFILW("OUT1",IER21)  
IF(IER21.NE.1)TYPE"ERROR on DFILW IER21= ",IER21
```

C

```
OPEN 4,"OUT1" ;recognition results .  
CALL DFILW("OUT2",IER21)  
IF(IER21.NE.1)TYPE"ERROR on DFILW IER21= ",IER21
```

C

```
OPEN 12,"OUT2" ;recognition results .  
CALL OPEN(1,"CORRAR",1,512,IER1)  
IF(IER1.NE.1)TYPE"ERROR IER1 =",IER1  
CALL CHECK (IER1)
```

C

```
OPEN 3,"SFACOR" ;scaling factor from PHDIST
```

C

C

C

```
CALL FGTIME(IHOUR,IMIN,ISEC)  
CALL DATE(IDATE,IER2)
```

C

```
CALL CHECK (IER2)
```

C

C

```
C***** Obtain and then write out to OUT(1) info  
C about the speechfile recognized.
```

C

C

```
ACCEPT" ENTER the words spoken ( < 50 char ):  
/ <15>"  
READ(11,15) ISAID(1)
```

```
15 FORMAT(S50)
```

C

```
ACCEPT" ENTER a speaker identifier (KS etc.)  
/ ( < 10 char ):<15>"  
READ(11,20) ISPKR(1)
```

```
20 FORMAT(S10)
```

C

```

C
C*****   Print a heading to file OUT1.  This file
C   contains recognition results only!!!
        WRITE (12,103) ISAID(1)
102      FORMAT (20X,"SENTENCE SPOKEN : ",S50 /)
103      FORMAT (S50)
C
        WRITE (12,105) ISPKR(1)
104      FORMAT (20X,"SPEAKER WAS : ",S10 /)
105      FORMAT (S10)
C
C
C
C
C*****   Now output an informative heading to explain
C   results of the distance measurement routine.
C
        WRITE(12,207)IDATE
207      FORMAT(15X,"THE DATE IS--"2I3,I5)
        WRITE(12,208)IHOURL,IMIN,ISEC
208      FORMAT(15X,"THE TIME IS--",3I3 //)
C
C*****   Write a header for the list's of output.
C
C
        WRITE(12,720)                                ;send the
        WRITE(12,722)                                ;header out
        WRITE(12,724)                                ;now.
C
720      FORMAT(4X,"VECTOR",3X,"FIRST",2X,"SECOND",3X,"THIRD",
/        2X,"FOURTH",3X,"FIFTH",9X,"SCALE")
C
722      FORMAT(4X,"NUMBER",2X,"CHOICE",2X,"CHOICE",2X,"CHOICE",
/        2X,"CHOICE",2X,"CHOICE",8X,"FACTOR")
C
724      FORMAT(4X,"*****",2X,"*****",2X,"*****",2X,"*****",
/        2X,"*****",2X,"*****",8X,"*****" /)
C
C
C
C
C
C   the following output makes use of
C   information contained in the three(15)"
C   files listed below.  This output(15)"
C   consists of the five 'best' choices(15)"
C   of a matching prototype phoneme(15)"
C   for each time frame along with a(15)"
C   a set of statistics on the phoneme(15)"
C   choices for each frame.
C   CORRAR - Contains spectrographic(15)"
C   distances between prototype phonemes(15)"
C   and the speechfile.  These are the(15)"

```

```

C   results of the DIST routine, and (15)"
C   consist of raw data only. (15)(15)"
C   SFACOR - Contains the scale factor(15)"
C   used to relate two identical scores(15)"
C   (say 100) in different time frames.(15)"
C   And the lower the value of this(15)"
C   number the better the phoneme match.(15)"
TYPE " "
TYPE "-Sentence length = length in vectors-"
TYPE "Remember that there are 4 vectors/block so "
TYPE "this program can recognize up to 100 blocks"
TYPE "of speech at one time "
TYPE "------(15)"

C
ACCEPT "STARTING BLOCK=",ISTART

C
C
C
C
ACCEPT "LENGTH OF SENTENCE (in vectors) =",IVECT
IVECT = IVECT - 1      ;adjust so don't overread
  NUMPP=71
  IEND=0
  IBEGIN=0
  IFLAG=0
  IFLAG1=0
  ICONT=1
  ISET=0
  ISENTL=IVECT
  IF (ISENTL.LE.200) GO TO 667
    ISENTL=200
    ICONT=2
666  DO 705 IC=1,ICONT
    IF (IC.EQ.1) GO TO 667
    REWIND 1
    REWIND 3
    ISENTL=IVECT-200
    ISTART=49+ISTART
    ISET=198
667  CONTINUE
    ISTBLK = 0 ;NUMBER OF BLOCK TO READ FIRST

C
C*****   Calculate number of first time increment used
C   in order to get the correct value out of RFACTOR.
C
C   ISAMP=((256.0/64) * ISTART) + 1      ;for 64 pt. DFT
C
C
C*****   The following loops read in the results of the
C   recognition routine (PHDIST),making sure that the
C   correct values of measurement distances are read
C   by setting up an offset as shown.
C
C*****   Note: The way in which the results are stored

```

```

C in CORRAR is as follows. Each two block grouping
C of data contains the distances between one phoneme
C (for blocks 0 & 1, the phoneme is the 1st phoneme;
C and for say, blocks 4 & 5, the phoneme is the 3rd
C phoneme) and each vector of the sentence.
C So for a speechfile that contains 360 vectors
C (= 90 Blocks of sampled data) of information
C the first 360 positions of each two block set
C mentioned above contains the first thru the
C 360th distance measure value between the phoneme
C specified by the two block grouping and each
C vector of the speech file.
C
C
C DO 500 IM=1,NUMPP
C   CALL RDBLK(1,ISTBLK,IBLK,2,IER3)
C   IF(IER3.NE.1) TYPE "ERROR ON RDBLK,IER3=",IER3
C   IF(IER3.EQ.9) TYPE "Check the number of phonemes used"
C   CALL CHECK (IER3)
C
C       ;Now read in distance measure results.
C
C   IOFFSET = ISAMP -1           ;offset used to
C                               ;get correct measure
C   DO 300 I=1,ISENTL           ;for each vector.
C   INVAL = I + IOFFSET         ;set up offset
C 300   ISTORE(I,IM) = IBLK(INVAL) ;get result.
C   ISTBLK = ISTBLK + 2         ;do next phoneme.
C
C 500   CONTINUE                ; END of input loop.....
C
C
C ***** For each time frame (vector) the following portion
C of this program will:
C
C   1. Scale the results of DIST routine
C      so each vector has a 0 to 100 relative
C      score on the best choice of a phoneme
C      for recognition purposes.
C
C   2. Determine the "best" five prototype
C      phoneme choices, and save the relative
C      score along with chosen phoneme.
C
C   3. Print out the "best" five choices
C      mentioned in 2, along with statistics
C      of the accuracy of the results.
C
C   ICH1VAL(J) = Value (0 to 100) of the first choice
C               phoneme for a given time frame.
C
C   ICH3PHO(J) = ID'FIER of the phoneme which is the third

```

```

C           choice for a given time frame (I).
C
C
C*****      Initialize the array's for the default choice.
C
      DO 550 K=1,ISENTL
      ICH1VAL(K)=0
      ICH2VAL(K)=0
      ICH3VAL(K)=0
      ICH4VAL(K)=0
      ICH5VAL(K)=0
C
      ICH1PHO(K)=SYMBOL(71)
      ICH2PHO(K)=SYMBOL(71)
      ICH3PHO(K)=SYMBOL(71)
      ICH4PHO(K)=SYMBOL(71)
      ICH5PHO(K)=SYMBOL(71)
C
C 550          CONTINUE
C
C
C*****      Determine the five "best" choices for
C each time frame.
C
      ISENT = ISENTL - 2
      DO 705 J=1,ISENT
      DO 700 I=1,NUMPP
      ISTORE(J,I) = IFIX(FLOAT(ISTORE(J,I))/32767 * 100)
C
C
C
C
      IF(ISTORE(J,I).GT.ICH1VAL(J)) GO TO 640
      IF(ISTORE(J,I).GT.ICH2VAL(J)) GO TO 635
      IF(ISTORE(J,I).GT.ICH3VAL(J)) GO TO 630
      IF(ISTORE(J,I).GT.ICH4VAL(J)) GO TO 625
      IF(ISTORE(J,I).GT.ICH5VAL(J)) GO TO 620
C
C
C 700          CONTINUE
C
C
C*****      NOTE: If start processing in the middle of
C a file, be sure that you start reading the correct
C SFACTOR values.
C
      IF(J.GT.1) GO TO 710
      IF(ISTART.EQ.0) GO TO 710
C
      ISAMP2 = ISAMP - 1
C
      DO 709 K01 = 1,ISAMP2
C

```

```

      READ (3,803) RDUMPSF      ;align SF file.
C
709      CONTINUE
C
710      CONTINUE
C
      READ (3,803) SFACTOR      ;SFACTOR (J)
C
803      FORMAT(F11.8)      ;factor I/O format stmt.
C
C
C
C*****      If start in the middle of a file adjust the
C      vector value printed...
C
      JVECTOR = J + ISAMP - 1
C
C
C
C
C*****      Now send out a listing of information for this
C      time frame.
C
C
      IF ((ICH1PHO(J).NE."01").AND.(ICH1PHO(J).NE."71")) IFLAG=1
      IF (IFLAG.EQ.0) GO TO 660
      IF (IFLAG1.EQ.1) GO TO 661
      IFLAG1=1
      IBEGIN=J+ISET
661      CONTINUE
      IF ((ICH1PHO(J).NE."01").AND.(ICH1PHO(J).NE."71")) IEND=J+ISET
660      CONTINUE
C
C
      WRITE(4,761) JVECTOR,
      + (ICH1PHO(J)),ICH1VAL(J), (ICH2PHO(J)),ICH2VAL(J),
      + (ICH3PHO(J)),ICH3VAL(J), (ICH4PHO(J)),ICH4VAL(J),
      + (ICH5PHO(J)),ICH5VAL(J),SFACTOR
C
C
C
C
C
761      FORMAT(5X,I4,3X,A2," ",I3,2X,A2," ",I3,2X,A2," ",I3,
      /      2X,A2," ",I3,2X,A2," ",I3,3X,F11.8)
C
C
C
705      CONTINUE
C
      GO TO 1000
C
C
C
C

```

```

;stop.

```

```

C***** The following subsets set the correct values
C of ICH-VAL and ICH-PHO depending on the results of
C the search for "best" five phoneme choices at each
C time frame.
C
620 ICH5VAL(J) = ISTORE(J,I) ;new score.
ICH5PHO(J) = SYMBOL(I) ;new phoneme #
GO TO 700 ;continue

C
C
625 ICH5VAL(J) = ICH4VAL(J) ;shift down
ICH5PHO(J) = ICH4PHO(J) ;scores and #'s.
ICH4VAL(J) = ISTORE(J,I) ;new score.
ICH4PHO(J) = SYMBOL(I) ;new #.
GO TO 700 ;continue.

C
C
630 ICH5VAL(J) = ICH4VAL(J) ;shift down
ICH5PHO(J) = ICH4PHO(J) ;scores and #'s.
ICH4VAL(J) = ICH3VAL(J) ;shift
ICH4PHO(J) = ICH3PHO(J) ;again.
ICH3VAL(J) = ISTORE(J,I) ;new score.
ICH3PHO(J) = SYMBOL(I) ;new #.
GO TO 700 ;continue

C
C
635 ICH5VAL(J) = ICH4VAL(J) ;shift
ICH5PHO(J) = ICH4PHO(J) ;
ICH4VAL(J) = ICH3VAL(J) ;shift
ICH4PHO(J) = ICH3PHO(J) ;
ICH3VAL(J) = ICH2VAL(J) ;shift
ICH3PHO(J) = ICH2PHO(J) ;
ICH2VAL(J) = ISTORE(J,I) ;new score.
ICH2PHO(J) = SYMBOL(I) ;new #.
GO TO 700 ;continue.

C
C
640 ICH5VAL(J) = ICH4VAL(J) ;shift
ICH5PHO(J) = ICH4PHO(J) ;
ICH4VAL(J) = ICH3VAL(J) ;shift
ICH4PHO(J) = ICH3PHO(J) ;
ICH3VAL(J) = ICH2VAL(J) ;shift
ICH3PHO(J) = ICH2PHO(J) ;
ICH2VAL(J) = ICH1VAL(J) ;shift
ICH2PHO(J) = ICH1PHO(J) ;
ICH1VAL(J) = ISTORE(J,I) ;new score.
ICH1PHO(J) = SYMBOL(I) ;new #.
GO TO 700 ;continue.

C
C
C***** Now send the recognition results (best choice
C for each vector) to the file OUT2. Be sure to
C send correct vector choice to the right file

```

```

C   position if adding to a file.....
C
C
C
1000   J=1
        CLOSE 1
        CLOSE 3
        REWIND 4
        DO 1121 ICNT=1,IBEGIN
      READ(4,1130) JVECTOR,
      + (ICH1PHO(J)),ICH1VAL(J), (ICH2PHO(J)),ICH2VAL(J),
      + (ICH3PHO(J)),ICH3VAL(J), (ICH4PHO(J)),ICH4VAL(J),
      + (ICH5PHO(J)),ICH5VAL(J),SFACTOR
1121   CONTINUE
        IGAP=IEND-IBEGIN+1
        IF (IGAP.LE.0) GO TO 1122
        DO 1122 ICNT=1,IGAP
      WRITE(12,1130) JVECTOR,
      + (ICH1PHO(J)),ICH1VAL(J), (ICH2PHO(J)),ICH2VAL(J),
      + (ICH3PHO(J)),ICH3VAL(J), (ICH4PHO(J)),ICH4VAL(J),
      + (ICH5PHO(J)),ICH5VAL(J),SFACTOR
      READ(4,1130) JVECTOR,
      + (ICH1PHO(J)),ICH1VAL(J), (ICH2PHO(J)),ICH2VAL(J),
      + (ICH3PHO(J)),ICH3VAL(J), (ICH4PHO(J)),ICH4VAL(J),
      + (ICH5PHO(J)),ICH5VAL(J),SFACTOR
1122   CONTINUE
1130   FORMAT(5X,I4,3X,A2," ",I3,2X,A2," ",I3,2X,A2," ",I3,
      /      2X,A2," ",I3,2X,A2," ",I3,3X,F11.8)
C
1131   FORMAT(I4,A2,I3,A2,I3,A2,I3,A2,I3,A2,I3,F11.8)
C
C
1140   CALL RESET
C
      TYPE"(15)<(15)<(15)"
      TYPE"----RECOGNITION RESULTS ALONE IN FILE 'OUT1'----"
      TYPE"----RESULTS WITH NOISE REMOVED IN FILE 'OUT2'----"
      TYPE"(15)<(15)"
      STOP----
      END

```

VITA

Gerard J. Montgomery was born on 10 February 1958 in Louisville, Kentucky. He graduated from Saint Xavier High School in 1976, and graduated from Speed Scientific School, University of Louisville, with the degree of Bachelor of Science in Applied Science in May 1980. Shortly after graduation he attended Officer Training School, and was commissioned in the United States Air Force.

Permanent address: 2819 Pomeroy Drive
Louisville, Kentucky 40218

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER GE/EE/82D-74	2. GOVT ACCESSION NO. A124851	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ISOLATED WORD RECOGNITION USING FUZZY SET THEORY		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
7. AUTHOR(s) Gerard J. Montgomery, Lt, USAF		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
12. REPORT DATE December 1982		13. NUMBER OF PAGES 283
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE		
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES 4 JAN 1983 Approved for public release; IAW AFR 190-17 Fredrick G. Lynch, Major, USAF, Director of PA Lyon E. Wolaver Dean for Research and Professional Development Air Force Institute of Technology (ATC) Wright-Patterson AFB OH 45433		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Speech recognition Continuous speech recognition Fuzzy Set Theory Fuzzy Algorithms Supervised learning		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A unique algorithm was developed to recognize isolated words given the output of an extremely variable feature extraction process. Because of the high error rate of the acoustic processor, it was necessary to rely on the consistency of the sequences of phonemes, and the errors that typically occur for a given word to determine the word spoken. This was accomplished by generating error statistics and phoneme representations for each word in the		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT

vocabulary using a set of training speech files. The top five, phoneme choices provided by the acoustic processor, and information indicating the accuracy of each choice, for each time segment of speech was implemented. Fuzzy set theory was used to combine this information with the error information obtained from the training files to determine the word spoken. Because of the number and types of errors present in the acoustic output, the algorithm produces a word score indicating the plausibility of a word being spoken regardless of the number of errors that occurred. To recognize a speech file, a score is generated for each word in the vocabulary, and the word with highest score is chosen as the word that was spoken. The number of operations required to make a decision increases linearly as the size of the vocabulary increases, and the algorithm can readily be adapted for real time speech recognition. The algorithm can also be applied to a large range of other problems where decisions must be based on data containing numerous errors.